

LOAN DOCUMENT

DTIC ACCESSION NUMBER	<div style="border: 1px solid black; width: 100px; height: 80px; margin: 0 auto;"></div> <p>LEVEL</p>	<p>PHOTOGRAPH THIS SHEET</p>	<div style="border: 1px solid black; width: 100px; height: 80px; margin: 0 auto;"></div> <p>INVENTORY</p>														
	<p>DOCUMENT IDENTIFICATION</p>																
	<p>DISTRIBUTION STATEMENT</p>																
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td colspan="2" style="text-align: center;">ACCESSION BY</td> </tr> <tr> <td style="width: 50%;"> NTIS GRAM DTIC TRAC UNANNOUNCED JUSTIFICATION </td> <td style="width: 50%; text-align: center;"> <input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> </td> </tr> <tr> <td colspan="2" style="height: 20px;">BY</td> </tr> <tr> <td colspan="2" style="height: 20px;">DISTRIBUTION/</td> </tr> <tr> <td colspan="2" style="text-align: center;">AVAILABILITY CODES</td> </tr> <tr> <td style="width: 30%;">DISTRIBUTION</td> <td>AVAILABILITY AND/OR SPECIAL</td> </tr> <tr> <td style="height: 60px; vertical-align: middle; text-align: center;">A-1</td> <td></td> </tr> </table>				ACCESSION BY		NTIS GRAM DTIC TRAC UNANNOUNCED JUSTIFICATION	<input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	BY		DISTRIBUTION/		AVAILABILITY CODES		DISTRIBUTION	AVAILABILITY AND/OR SPECIAL	A-1	
ACCESSION BY																	
NTIS GRAM DTIC TRAC UNANNOUNCED JUSTIFICATION	<input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>																
BY																	
DISTRIBUTION/																	
AVAILABILITY CODES																	
DISTRIBUTION	AVAILABILITY AND/OR SPECIAL																
A-1																	
<p>DISTRIBUTION STAMP</p>		<p>DATE ACCESSIONED</p>															
<p>DATE RECEIVED IN DTIC</p>		<p>DATE RETURNED</p>															
<p>19981223 071</p>		<p>REGISTERED OR CERTIFIED NUMBER</p>															
<p>PHOTOGRAPH THIS SHEET AND RETURN TO DTIC-FDAC</p>																	

HANDLE WITH CARE

UNCLASSIFIED

NO DISTRIBUTION
STATEMENT

NADC

Tech. Info.

APPENDIX 13

EMITTER CLASSIFICATION PROCESSING

FINAL SOFTWARE REPORT

DATA ITEM NO. A005

Reproduced From
Best Available Copy

**INTEGRATED ELECTRONIC WARFARE SYSTEM
ADVANCED DEVELOPMENT MODEL (ADM)**

7800987-14
PREPARED FOR:
NAVAL AIR DEVELOPMENT CENTER
WARMINSTER, PENNSYLVANIA
CONTRACT #62269-75-C-070

RAYTHEON
ELECTROMAGNETIC
SYSTEMS DIVISION

1 OCTOBER 1977

UNCLASSIFIED

APPENDIX 13
EMITTER CLASSIFICATION PROCESSING DESIGN SPECIFICATION
FINAL SOFTWARE REPORT
DATA ITEM A005

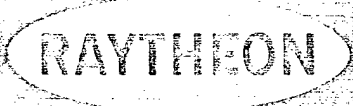
INTEGRATED ELECTRONIC WARFARE SYSTEM (IEWS)
ADVANCED DEVELOPMENT MODEL (ADM)

Contract No. N62269-75-C-0070

Prepared for:
Naval Air Development Center
Warminster, Pennsylvania

Prepared by:
RAYTHEON COMPANY
Electromagnetic Systems Division
6380 Hollister Avenue
Goleta, California 93017

1 OCTOBER 1977

		RAYTHEON COMPANY		LEXINGTON, MASS. 02173		CODE IDENT NO. 49956		SPEC NO. 53959-GT-0760							
						SHEET 1 OF 89		REV							
TYPE OF SPEC COMPUTER SUBPROGRAM DESIGN DOCUMENT															
TITLE OF SPEC EMITTER CLASSIFICATION PROCESSING - 1															
FUNCTION		APPROVED		DATE		FUNCTION		APPROVED		DATE					
WRITER		<i>[Signature]</i>		12/7/76											
		<i>[Signature]</i>		12/7/76											
REVISIONS															
CHK		DESCRIPTION				REV		CHK		DESCRIPTION				REV	
REVISION															
SHEET NO.															
REV STATUS OF SHEETS		REVISION													
		SHEET NO.													

RAYTHEON

RAYTHEON COMPANY

LEXINGTON, MASS. 02173

CODE IDENT NO.

49956

SPEC NO.

53959-GT-0760

SHEET

2 OF 89

REV

Table of Contents

Section:	Title	Page
1.0	SCOPE	3
1.1	Identification	3
1.2	Subprogram Tasks	4
2.0	REFERENCE DOCUMENTS	5
2.1	Performance Specification	5
2.2	Program Design Specification	5
3.0	REQUIREMENTS	6
3.1	Subprogram Detailed Description	6
3.1.1	ECDR-Emitter Classification Driver	6
3.1.2	ECLV1- Emitter Classification Level 1	8
3.1.3	ECST1 - Scan Test 1	26
3.2	Flowcharts	28
3.3	Computer Subprogram Environment	47
3.3.1	Local Permanent Tables	48
3.3.2	Local Ephemeral Storage	58
3.3.3	Common Data Base References	69
3.4	Input/Output Formats	70
3.4.1	ECDR-Driver	70
3.4.2	ECLV1 (Subroutine of ECDR)	71
3.4.3	ECST1 (Subroutine of ECDR)	74
3.5	Required External Subroutines	75
3.6	Conditions for Initiation	76
3.7	Subprogram Limitations	77
3.7.1	Timing	77
3.7.2	Algorithm Limits	77
3.7.3	Error Testing/Reasonableness Checks	77
3.8	Interface Description	78
Appendix A	Binary Search (BIRCH)	81
Appendix B	Intersection Routine (INSECT)	84
Appendix C	Parity - Counting One's Bits	89

1.0 SCOPE

1.1 IDENTIFICATION

This subprogram shall be known by the designator ECDR (Emitter Classification Driver).

Designators and brief functional descriptions of subroutines written specifically for use by ECDR are shown in Table 1.

Table 1

ECDR Subroutines - Designators & Function

Designator	Brief Functional Description
ECLV1	Emitter Classification Level 1 - Matches frequency, PRI and PW of input emitter track file to limits of emitter library 1 and returns with no match indication or else a candidate list. Called by ECDR.
BIRCH	<u>Binary Search</u> subroutine. Called by ECLV1.
TRNSL8	<u>Translates</u> candidate list in keyword - compressed format to standard candidate list format. Called by ECLV1.
INSECT	Generates the <u>intersection</u> of 2 or 3 candidate trunk lists in keyword - compressed format. Result in same format. Called by ECLV1 and TRNSL8.
PARITY	Generates the 1's bit count of A-Reg in A-Reg. Called by INSECT.
ECST1	Scan Test 1: Requests scan analysis if amplitude threshold exceeded. Else calls SCTCOM.

A seventh subroutine SCTCOM is shared by Scan Tests 1 and 2. It complements the ETF state indicator and if this is now = 1, sets scan type = circular and scan period = time out.

2.0 REFERENCE DOCUMENTS

2.1 PERFORMANCE SPECIFICATION

Computer Program Performance Specification (CPDS), IEWS, ADM (U), Raytheon Company, ESD, (No. 061290529), (1 June 1976), (U).

2.2 PROGRAM DESIGN SPECIFICATION

System Controller Program, IEWS, ADM - - CPDS Number 53959-GT-0750, 1976 September 2.

2.3 DATA BASE DESIGN DOCUMENT

Common Data Base Design Document (CDBDD), SC Unit, IEWS, ADM,
Raytheon Company, ESD, (No. 53959-GT-0751), (13 October 1976), (U).

2.4 MISCELLANEOUS DOCUMENTS

Document No.

Document Title

53959-GT-0756

Computer Subprogram Design
Document, Executive, IEWS, ADM

53959-GT-0759

Computer Subprogram Design
Document, Data Extraction, IEWS, ADM

3.0 REQUIREMENTS

3.1 SUBPROGRAM DETAILED DESCRIPTION

3.1.1 ECDR - - Emitter Classification Driver

ECDR is the logical skeleton for emitter classification processing, first level. It accomplishes its work through two local subroutines:

ECLV1 - Emitter Classification Level 1, described hereafter (Section 3.1.2).

ECST1 - Emitter Classification Scan Test 1, described hereafter (Section 3.1.3).

both of which have dependent subroutines of their own; and one external, executive level subroutine EXMES. See flow diagram 3.2.0.

ECDR calls the two local subroutines cited, interprets their results, formats the scan-analysis request message, and sends this message to the Executive. A step-by-step account follows. The steps are either program labels or unlabeled sections headed by a semi-colon followed by a number.

ECDR - Save A-Reg. ECDR uses only A and X-Reg's; other registers are saved and restored by ECDR's dependent subroutines.

Upon entry, X→first word of a 3-word block. The third word contains the EFN (Emitter Track File #) as a whole-word item. This is fetched to the A-Reg, moved to the X-Reg, and saved on the stack.

```
;2 - Add the EC Return Module code (stored in the left byte
      as parameter RMCDLB) into the A-Reg (which still has
      EFN) to create a byte-split word:
```

Return Module Code	EFN
-----------------------	-----

Store this in the Scan-Analysis request message block at label SRRMCD. See Section 3.3.1.4.

- ;3 - Call ECLV1. The input is X-Reg = EFN done above. A return to Call +1 implies that no Candidate List was created. The Call +1 location is a JMP to step ECEND below. Taking this path means returning from ECDR without sending any message to the Executive.

A return to Call +2 is to next step.

- ;4 - The X-Reg now points to the 1st word of the Candidate List. This is stored in the Scan-Analysis request message at label SRCLAD. See Section 3.3.1.4.

- ;5 - The X-Reg is reloaded with EFN (Non-destructively from top of stack). This is required as input for the forthcoming call on ECST1.

The A-Reg is loaded with the constant X'8000'. This will leave the AW bit of the Scan-Analysis request message set if ECST1 returns to Call +2.

Call ECST1. A return to Call +1 implies no scan analysis is required. The Call +1 location contains a clear instruction that resets the AW bit that was set above.

A return to Call +2 is to next step.

- ;6 - The A-Reg now contains either 0000 (No analysis) or X'8000' (Analysis wanted). This is now increased by 1 to set the type of analysis wanted or not wanted to "SCAN". The completed request word is stored in the Scan-Analysis request message at label SRREQW. See Section 3.3.1.4.

- ;7 - The X-Reg is loaded with the address of the 1st word of the now completed Scan-Analysis request message, and the message is sent by a call to EXMES.

ECEND - This step is reached from Step 7 or from Step 3 if ECLV1 returns to Call +1.

Load X-Reg from top of stack. This is not the input value of X-Reg, but = EFN.

Restore A-Reg from stack.

Return.

3.1.2 ECLV1 - Emitter Classification Level 1

This is the workhorse subroutine of ECDR. Its overall function is to create a list of Candidate Emitters each of which could match the input Emitter Track file in the three measurements: frequency, PRI and PW. The steps are as follows:

ECLV1 - Save A, B and E-Reg's on stack. Store the (local) address, CANDLE, of the Candidate List to be generated on the stack as the exit value of the X-Reg. Save the EFN, which is now in X-Reg in the first word of the Candidate List.

;1 - Make room on the stack for a block of 3 words which will later be used to hold the addresses of the Parameter-Matching Trunk lists as they are found. Push the address of the first word of this block to the stack. Symbolic name of this address is ala.⁽¹⁾

;2 - Make room on the stack for nvps⁽¹⁾, the number of valid parameters and an un-named block of 17 words to be used to hold the intersection of 2 or 3 Parameter-Matching Trunk lists. Store the address of the first word of this block on the stack. Symbolic name of this address is adint.⁽¹⁾

;3 - Create in X-Reg the address of ETF (EFN) = $ETF + 16 * EFN$. This stays in X-Reg until the end of the subroutine, and is used to locate various items in the ETF.

;4 - Set nvps⁽¹⁾ = \emptyset .

LOOKFR - Fetch the word (EFVD rel to X) containing the frequency validity bit, mask it (Mask = EFVF), and test it for zero. If it = \emptyset , there is no valid frequency measurement: Go to step LOOKPI.

⁽¹⁾ We are using lower case letters to distinguish between the item (lower case) and the symbolic displacement (upper case) used with base S to access the item. See Section 3.3.2.1.

3.1.2 continued -

- ;5 - Fetch frequency (EFRQD rel to X) to A-Reg as Binary Search key, and call internal subroutine LOOK. 1 with these three arguments stored in order after the JSB: ODV. F, ODA. F and ZLS. F. See LOOK. 1 below.

If the search is unsuccessful, control will go directly to Step FAIL. 1, else LOOK. 1 will return and proceed to next step.

LOOKPI - This step is reached from Steps LOOKFR and 5.

Fetch the word (EPIVD rel to X) containing the PRI validity bit, mask it (Mask = EPIVM), and test it for zero. If it = \emptyset , there is no valid PRI measurement: Go to Step LOOKPW.

- ;6 - Fetch average PRI (EAPID Rel to X) to A-Reg as Binary Search key (masking it with Mask = EAPIN) and call internal subroutine LOOK. 1 with these three arguments stored in order after the JSB: ODV. PI, ODA. PI and ZLS. PI. See LOOK. 1 below.

If the search is unsuccessful, control will go directly to Step FAIL. 1, else LOOK. 1 will return and proceed to next step.

LOOKPW - This step is reached from Steps LOOKPI and 6.

Fetch the word (EPWVD rel to X) containing the PW validity bit, mask it (Mask = EPWVM), and test it for zero. If it is = \emptyset , there is no valid PW measurement: Go to Step INSPCT.

- ;7 - Left concatenate long pulse flag (ELPD rel to X), using mask ELPM and shift count LP.RS to pulsewidth field (EPWD rel to X-Mask EPWM) to form a direct ODA-Look up index. Leave this in B-Reg.
- ;8 - Call internal subroutine LOOK. 2 with these two arguments stored in order after the JSB: ODA. PW, ZLS. PW. See LOOK. 2 below.

If the search is unsuccessful, control will go directly to Step FAIL. 1, else LOOK. 2 will return and proceed to next step.

3.1.2 continued -

;9 - Put address of result (CANDLE) in B-Reg, and call TRNSL8. This will translate the list whose address is in A-Reg from KWC/BV format to standard (CDBDD) Candidate List format, and store result in block pointed to by B-Reg on input. See Section 3.3.1.1.

NORMAL - A valid Candidate List has been produced. ECLV1 communicates this to its caller by returning to Call +2. In this step, bump return address to Call +2 by $rtad \leftarrow rtad + 1$.

ABNORM - This step is reached from step NORMAL and from Step 11 which is the terminal step of the common sequel to the two failure steps FAIL.1 and FAIL.2.

Clean up stack, i.e., return to available status the 23 locations appropriated for local temporary storage in entry Steps 1 and 2. This is done by adding 23 to S-Reg. S-Reg then points to the exit value (CANDLE) of X-Reg.

Restore X, E, B and A-Reg's from stack and exit to Call +2 or Call +1 as Step NORMAL was or was not executed.

** - This ends the flow of ECLV1. We must, however, give account of the two failure steps FAIL.1 and FAIL.2 and the two internal subroutines LOOK.1 and LOOK.2.

FAIL.1 - This step is reached from internal subroutines LOOK.1/2 on no-match finding in any of the three calls on these subroutines (from Steps 5, 6, and 7), and from Step ISECT if the 2 or 3-way intersection proves to be null ($A\text{-Reg} = \emptyset$ on return).

In this step put the failure code 'NOFA1' (None-of-the-above - - 1) in the B-Reg and go to Step 10.

FAIL.2 - This step is reached only from Step INSPCT when $nvps = \emptyset$.

Put the failure code 'UNCLS' (Unclassified) in the B-Reg and go to Step 10.

3.1.2 continued -

- ;10 - Store failure code that is in B-Reg in the ETF Identity field (EIDD Rel to X), leaving other fields unchanged (Mask EIDM).
- ;11 - Store "Unknown" display code ('UNKNO') in ETF display code field (EDISD Rel to X), leaving other fields unchanged (Mask EDISM).

To to Step ABNORM.

LOOK.1 - Pop top of stack into E-Reg. This leaves E pointing at the word following the JSB, i.e., at the ODV argument.

Load ODV argument into B-Reg and set $E \leftarrow E + 1$ (i.e., $E \rightarrow$ now to ODA argument).

Call BIRCH (Binary Search). The arguments are A-Reg = Search Key and B-Reg → Search List. A-Reg was set before the call LOOK.1 in Steps 5 and 6. Upon return B-Reg contains an index into ODA.

Go to Step 1KCOM.

LOOK.2 - Pop top of stack into E-Reg. This leaves E pointing to the word following the JSB, i.e., at argument ODA.PW. The B-Reg was set to the index into ODA.PW prior to entry (Step 7).

Go to Step LKCOM.

LKCOM - This step is reached from Steps LOOK.1 and LOOK.2. In both cases, one reaches this step with B-Reg = Index into ODA List and E-Reg → address of ODA argument.

Add ODA argument to B-Reg (Index) and set $E \leftarrow E + 1$ (i. e., $E \rightarrow$ now to ZLS argument).

Pick up ODA at entry whose address was just been constructed in B-Reg. Call this D(L).

If $D(L) < \emptyset$, the parameter we are currently looking at matches none of the ELL1 emitters: Go to Step FAIL.1, else next step.

3.1.2 continued -

;12 - Add ZLS argument to D(L) to form a ZLS-subsection address, and set $E \leftarrow E + 1$ (i.e., $E \leftarrow$ now to return point).

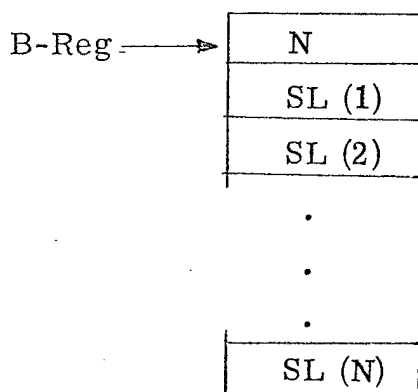
Store the address thus created at address $ala + nvps$.

Set $nvps \leftarrow nvps + 1$.

Return to location pointed to by E-Reg.

3.1.2.1 BIRCH - Binary Search Subroutine

This is a subroutine of ECLV1. Its required inputs are A-Reg = Search Key (SK) and B-Reg = Address of Search List (SL):



where $SL (1) < SL (2) < \dots < SL (N)$ and it is assumed that $SL (1)$ is less than any value actually taken by SK.

The version of Binary Search used here is Algorithm U, Section 6.2.1, D. C. Knuth, The Art of Computer Programming, Vol. 3, P. 411.

The steps are as follows:

BIRCH - Save A, X and E-Reg's on stack.

;2 - Set X-Reg = -1 for right-shift-1 count.

;3 - Set E-Reg = -N and $B \leftarrow B + 1$ (i.e., B now \rightarrow SL (1)).
Set A-Reg = 1's complement of $N \equiv C1 (N)$. Save
B-Reg = Addr [SL (1)] on stack.

3.1.2.1 continued -

LOOPG - This step is reached from Step 3 (First Loop Entry) and from Step GRTR, that is after a non-terminal comparison in which $SK >$ current SL comparand.

At this point, $E\text{-Reg} = -M$ where M is the termination criterion (Stop when M reaches \emptyset unless exact match found first).

Shift the register pair (X, E) right 1 bit. This keeps $X = -1$ and $E \leftarrow -(M + 1)/2$.⁽¹⁾

Go to COMON.

LOOPL - This step is reached only from Step LESS, i.e., after a non-terminal comparison in which $SK <$ current SL comparand.

The condition of E-Reg is as described in LOOPG.

Perform the same shift as in LOOPG, then change the sign of E-Reg so that $E = (M + 1)/2$.⁽¹⁾

Go to Step COMON.

COMON - This step is reached from LOOPG with $E = -(M + 1)/2$ and LOOPL with $E = +(M + 1)/2$. In either case E contains - Delta where Delta is the change to the SL Lookup address (in B-Reg) required to find the next comparand.

Alter SL Lookup address in B-Reg by $B \leftarrow B - E$

B-Reg is increased if in last comparison SK was $>$ last SL comparand and conversely.

Move $C1(M)$ from A-Reg to E-Reg and right shift register pair (X, E) one bit. E-Reg now contains $C1(M/2) = C1(\text{New } M)$.⁽¹⁾

(1) Note: All divisions in this algorithm are truncated integer divisions as in FORTRAN.

3.1.2.2 continued -

CLHIB - This step begins a loop. It is entered first time from Step 1 and loop-entry-value-of-B-Reg-minus-one times from the end of CLHIB itself. On entry, $X = \text{ODA.GT} + 1$, $B = \text{ngrps}$ so that $B + \bar{X} - 1$ is address of last ODA.GT entry. The loop is accomplished by diminishing B by 1 each iteration until it reaches \emptyset . Thus, X is unchanged by loop. The purpose of the loop is to clear the LSB of (ODA.GT + 1), (ODA.GT + ngrps) -- the "Hit-Bit".

Fetch $(X + B - 1)$ to A-Reg AND with $\bar{X}'\text{FFFE}'$, store result back at $(X + B - 1)$, $B \leftarrow B - 1$.

If $B \neq \emptyset$ go to Step CLHIB, else next.

GROOP - This step begins the major processing loop of the subroutine. It is entered first time from above and $N - 1$ times from Step TALLY, where $32 \leq N \leq \text{original value stored in ngrps}$. The lower limit, 32, is imposed by fact that there is room in the Candidate List output block for only 32 Candidates. On 1st entry to this step, $X\text{-Reg} = \text{ODA.GT} + 1$.

$\text{grpno} \leftarrow \text{grpno} + 1$

Pick up $((X)) \equiv (\text{ODA.GT} + I) \equiv D(I)$; $I \geq 1$ and simultaneously, $X \leftarrow X + 1$.

Arithmetic right shift $D(I)$ 1 bit to get rid of the Hit-Bit (LSB) in this working version of $D(I)$. The bit remains in the stored version of $D(I)$.

Test $D(I)$: Of $D(I) \geq \emptyset$ go to Step ISECT else next.

;2 - $D(I) < \emptyset$: This means that $D(I) = J - (I + 1)$ for some J such that $1 \leq J \leq I$. At this point, $(X\text{-Reg}) = \text{ODA.GT} + I + 1$ (Because of $X \leftarrow X + 1$ in Step GROOP) so that the address

$$(X) + D(I) = \text{ODA.GT} + I + 1 + J - (I + 1) = \text{ODA.GT} + J$$

is the address of $D(J) \geq \emptyset$ which is a ZLS.GT entry displacement previously considered in Step ISECT. The results of that consideration are recorded (see Step 3) in the LSB of $D(J)$ (Hit-Bit).

3.1.2.2 continued -

GOTHIT - This step is entered from Step 3 if the Trunk List of the PRESENT group had a non-null intersection with the input Candidate Trunk List- OR - from Step 2 if the present group's trunk list is identical to some previous (lower numbered) group's trunk list and the latter list had a non-null intersection, etc. In either case, the present group's number is grpno and the purpose of this step and Steps 4-6 is to record grpno in the output Candidate List.

Compute and leave in B-Reg the address of the EL2 file pertaining to grpno:

$$B \leftarrow EL2 + 11 * (\text{grpno} - 1) \text{ (call E2ADR)}$$

- ;4 - Look up the Group identity field (right byte of MIDth word relative to B), and build in E-Reg byte-split word:

Group Ident	<u>grpno</u>
-------------	--------------

- ;5 - Augment Candidate count:

$$\text{ncand} \leftarrow \text{ncand} + 1$$

and store byte-split result of Step 4 at address resad + ncand.

- ;6 - Test to see whether there is room in the output Candidate List to store any more Candidates: limit \leftarrow limit - 1

If limit $\neq 0$ go to Step TALLY. Else go to Step TRDUN.

TRDUN - This step is entered from either step.

6 : No room left in Output Candidate List

-OR-

TALLY: No more Groups left to consider

Whichever occurs first.

Store ncand in left byte of Output Candidate List header (1st word), i.e., at address resad. Note that

$$1 \leq \text{ncand} \leq 32,$$

3.1.2.2 continued -

TRDUN - i.e., we do not enter with a null Candidate Trunk List and each Trunk therein is associated with at least one Group.

;7 - Clean up stack, i.e., return to available status the temporary storage locations on stack appropriated in Steps TRNSL8 and 1 ($S\text{-Reg} \leftarrow S\text{-Reg} + 26$).

Return.

3.1.2.3 INSECT - Intersection Subroutine

This subroutine is called both by ECLV1 and by TRNSL8. The required inputs are:

A-Reg = Address of a list of 2 or 3 Input List addresses.

B-Reg = Address where intersection result is to be stored.

E-Reg = Switch instruction

= NOP if 2-way intersection

= JMP \$ +2 if 3-way intersection

Both input lists and output list are in KWC/BV format. (See Section 3.3.1.2.2). The steps of INSECT are as follows:

INSECT - Save X-Reg on stack. Save E and B Registers on stack in local temporary storage area with symbolic names instr and outad respectively.

;1 - Repeat the following (*) three (3) times:

* Fetch Input List address pointed to by A-Reg and $A \leftarrow A + 1$. Push Input List address to stack.

The symbolic names of these three addresses are adr1, adr2, and adr3.

Put the Absolute address of adr3 in X-Reg. Make room for eleven more temporary storage locations on stack ($S\text{-Reg} \leftarrow S\text{-Reg} - 11$). See map in 3.3.2.3.

3.1.2.3 continued -

- ;2 - Set A-Reg = \emptyset and B-Reg = X'FFFF'. A will be the head (most significant half) of the double words created in next 5 steps. B will be used to hold the keyword intersection KWI as it is developed.
- ;3 - Execute (E X Q) the Switch instruction instr. If it is a NOP control goes to the instruction following the EXQ and this is a jump to Step INS01. If instr is a JMP \$ + 2, go to next step.
- ;4 - X-Reg was set to address of adr3 in Step 1. Load E-Reg with KW3 = (adr3) and $X \leftarrow X + 1$. Store double register (A, E) = (\emptyset , KW3) as dw3. $B \leftarrow B \wedge KW3$. Go to Step 5.
- INS01 - This step is entered only from Step 3 and Step 4 has not been executed. Therefore advance X-Reg so that it points to adr2 ($X \leftarrow X + 1$).
- ;5 - Load E-Reg with KW2 = (adr2) and $X \leftarrow X + 1$. Store double register (A, E) = (\emptyset , KW2) as dw2. $B \leftarrow B \wedge KW2$.
- Load E-Reg with KW1 = (adr1) and $X \leftarrow X + 1$. Store double register (A, E) = (\emptyset , KW1) as dw1. $B \leftarrow B \wedge KW1$.
- ;6 - Define B-Reg contents = KWI (Keyword intersection). If KWI = \emptyset we have an early detection of null intersection: Go to Step INDUN1.
- Else move KWI to E-Reg and store double register (A, E) = (\emptyset , KWI) as dwi.
- ;7 - $rsi \leftarrow \emptyset$ (Result storage index)
 $cnt \leftarrow 16$ (Shift counter)

INLOOP - This major loop is entered as many times as there are 1's bits in KWI, initially from Step 7 and thereafter from Step INS05. All steps down to and including INS05 are inside this loop.

Set X-Reg = 1 and load double register (A, E) \leftarrow dwi.

3.1.2.3 continued -

INSHFT - Here begins a minor loop whose object is to shift (A, E) left one bit at a time until the LSB of A-Reg = 1 and to record the number of such shifts in X-Reg.

If E-Reg ≤ 0 (MSB (E) = 1) go to Step INS02.

Else, shift (A, E) left 1 bit (0 enters LSB (E))

$X \leftarrow X + 1$

Go to Step INSHFT

INS02 - Shift (A, E) left once more and store (A, E) as updated dwi. The shift count is in X-Reg and is maintained there through rest of loop. For ease of description we call it NSH from now on.

;11 - (A, E) \leftarrow dw1.

Shift (A, E) left NSH bits (NSH 0's enter at right)
dw1 \leftarrow (A, E).

Call PARITY to do A-Reg \leftarrow 1's bit Count (A-Reg), and add this to address adr1. We now have the List -1 binary vector word corresponding to the 1 in KWI which was just shifted into LSB [Head (dwi)]

Fetch this word and store as result word (rew).

;12 - Repeat Step 11 VIS-A-VIS dw2 except that word fetched from List -2 is ANDED with rew and held in A-Reg.

;13 - Execute (EXQ) switch instruction instr. As before, A NOP (2-way) takes us to a jump around next step (to Step INS03) while a JMP \$ + 2 (3-way) takes us to next step.

;14 - Store partial result in A-Reg as rew. Repeat Step 12 VIS-A-VIS dw3.

INS03 - A-Reg now contains the intersection of 2 or 3 binary vector words.

If A-Reg $\neq 0$, A-Reg is a new, valid binary vector word of the result: Go to Step INS04. else next step.

3.1.2.3 continued -

;16 - The binary vector words corresponding to the 1 in KWI which is now in the LSB of the head of dwi have a null intersection which is not to be stored in the result. Therefore -

Reset LSB [Head (dwi)] = \emptyset . Go to Step INS05.

INS04 - Do rsi \leftarrow rsi + 1

Create address outad + rsi and store result word there.

INS05 - Tests for completion. This step is reached from Steps 16 and INS04.

Do cnt \leftarrow cnt - NSH

If cnt now = \emptyset go to Step INDUN2

Else, do (A, E) \leftarrow dwi.

If TAIL (dwi), i. e., E-Reg $\neq \emptyset$, there are still 1's in KWI that have to be examined: Go to Step INLOOP else out of loop to next step.

;19 - Out of major loop. cnt was not = \emptyset but there were no more 1's left in KWI to be examined. dwi is in (A, E).

Shift (A, E) left cnt bits.

INDUN1 - This step is reached from Steps 6 (KWI = \emptyset to start, A-Reg = \emptyset), 19 (dwi is in (A, E) and cnt has been forced to \emptyset) and from step INDUN2 (cnt was = \emptyset , INDUN2 loaded A with head (dwi)).

In any case, A-Reg = true keyword of the result. Store this at address = (outad).

Clean up stack, i. e., return to available status the 16 words of temporary stack storage appropriated in entry steps INSECT and 1. This is done by S-Reg \leftarrow S-Reg + 16.

Restore X-Reg from stack.

Return to caller at Call + 1.

3.1.2.3 Continued -

INDUN2 - This step is entered only from Step INSØ5 when $\text{cnt} = \emptyset$. This can only come about if LSB of original KWI (Step 6) was = 1.

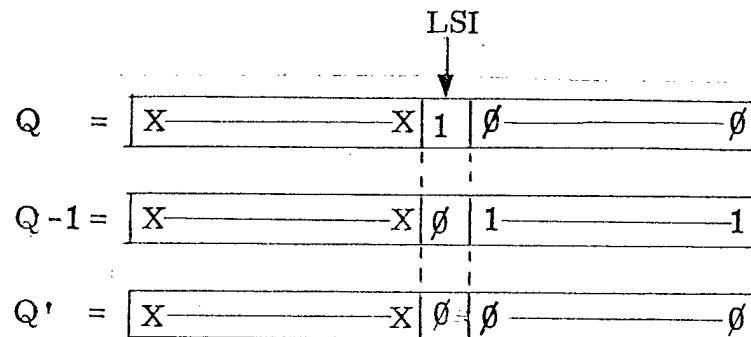
Set up proper exit conditions: Load A-Reg with head (dwi).

3.1.2.3.1 PARITY - Count 1's in A-Reg

This is a subroutine of INSECT. It is based on the following theorem:

Theorem: Let Q be an unsigned, non-zero binary number represented in a register of finite length. Then $Q' = Q \wedge (Q - 1)$ has one fewer 1 bit than Q .

Proof: Theorem is obviously true for $Q = 1$. Therefore, consider some $Q > 1$ and define LSI as the position of Q 's least significant 1 bit.



Input: Argument and result are both in A-Reg.

Steps:

PARITY - If A-Reg = Ø go to Step DONE.

;1 - Save B & E-Reg's on stack

Set $B \leftarrow \emptyset$. B-Reg will hold count until end.

PLOOP - $E \leftarrow A$
 $E \leftarrow E - 1$
 $A \leftarrow A \wedge E$
 $B \leftarrow B + 1$

RAYTHEON

RAYTHEON COMPANY
LEXINGTON, MASS. 02173

CODE IDENT NO.

49956

SPEC NO.

53959-GT-0760

SHEET
25 OF

REV

3.1.2.3.1 continued -

- ;2 - If A-Reg $\neq \emptyset$ go to Step PLOOP, else next step.
- ;3 - Move count from B-Reg to A-Reg. Restore E and B-Reg's from stack.
- DONE - This step reached from Step PARITY (input argument was = \emptyset) and Step 3 (algorithm complete).

In either case, A-Reg = result.

Return to caller at Call +1.

RAYTHEON

RAYTHEON COMPANY
LEXINGTON, MASS. 02173

CODE IDENT NO.

49956

SPEC NO.

53959-GT-0760

SHEET
27 OF

REV

3.1.3.1 continued -

SCTCOM - Bump return address to Call +2.

;1 - $B \leftarrow \text{State Indicator Mask (ESINM)}$

$A \leftarrow \text{State Indicator (ESIND rel to X)}$

$A \leftarrow \text{XOR (A, B)}$ - This complements State Indicator.
Store result (ESIND rel to X).

;2 - Isolate present value of State Indicator in A-Reg by
 $A \leftarrow A \wedge B$.

If State Indicator (A-Reg) = \emptyset go to Step OUT, else
next step.

;3 - Fetch statically encoded word containing both the circular
scan type (CIRC) and the time-out scan period (TOUT)
aligned to their respective fields. Store this in the common
location of the two fields (ESTYD rel to X).

OUT - Reached from Steps 2 and 3.

Return to Call +2.

RAYTHEON

RAYTHEON COMPANY
LEXINGTON, MASS. 02173

CODE IDENT NO.

49956

SPEC NO.

53959-GT-0760

SHEET
28 OF 89

REV

3.2 FLOWCHARTS

3.2.0 Emitter Classification Processing -1

3.2.1 ECDR (page 29)

3.2.2 ECLV1 (page 30)

3.2.2.1 BIRCH (page 37)

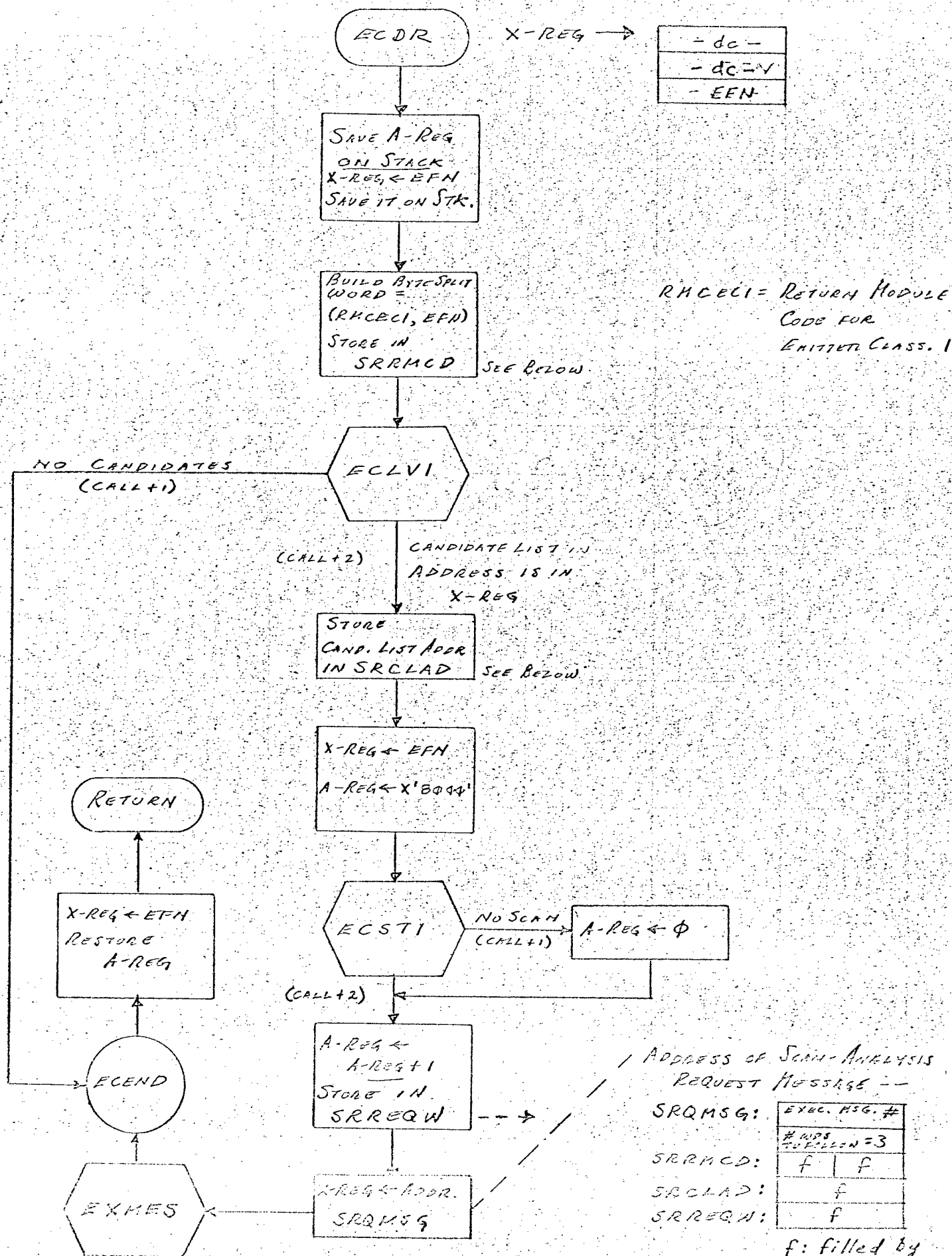
3.2.2.2 TRNSL8 (page 38)

3.2.2.3 INSECT (page 41)

3.2.2.3.1 PARITY (page 44)

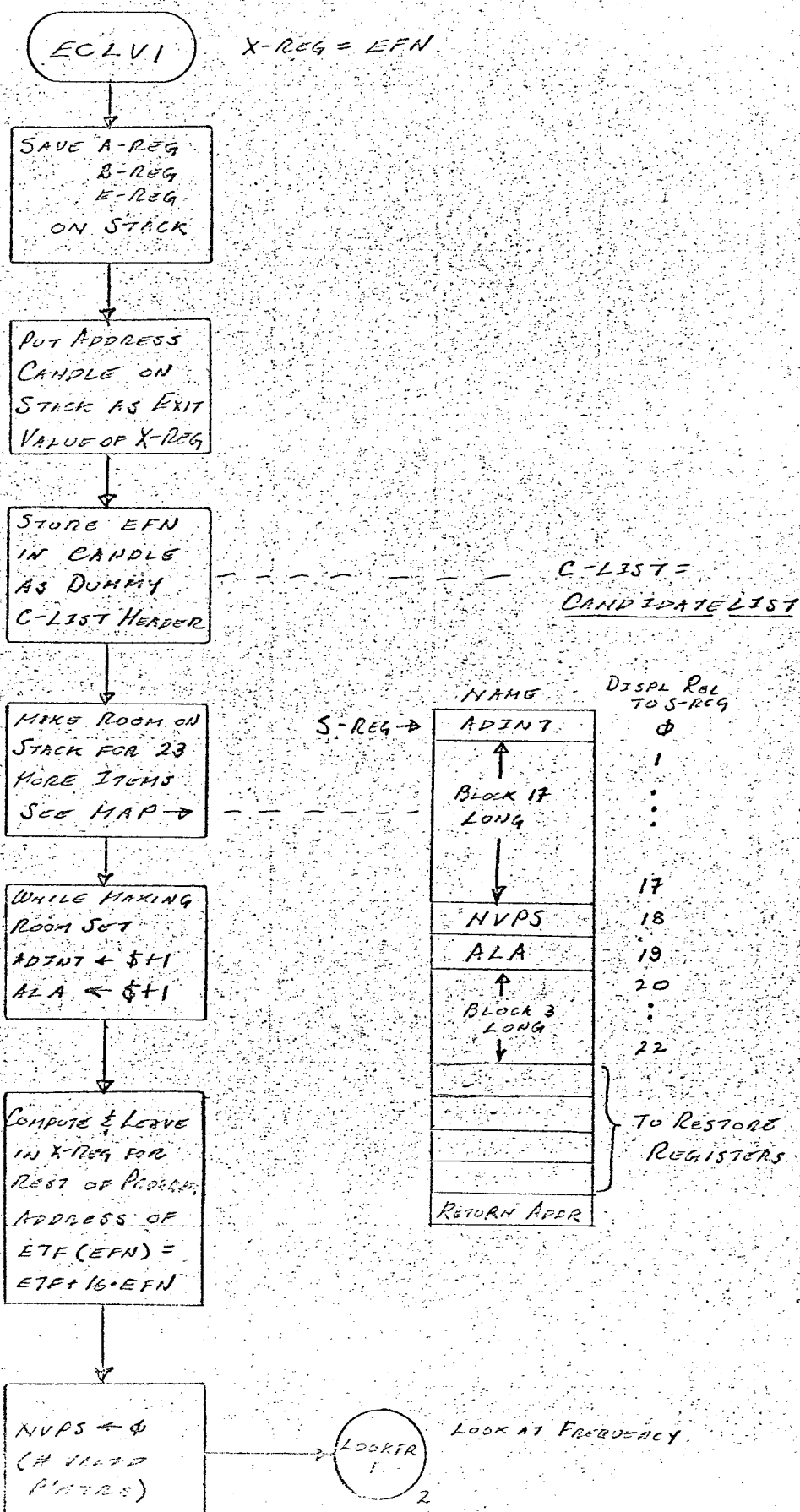
3.2.3 ECST1 (page 45)

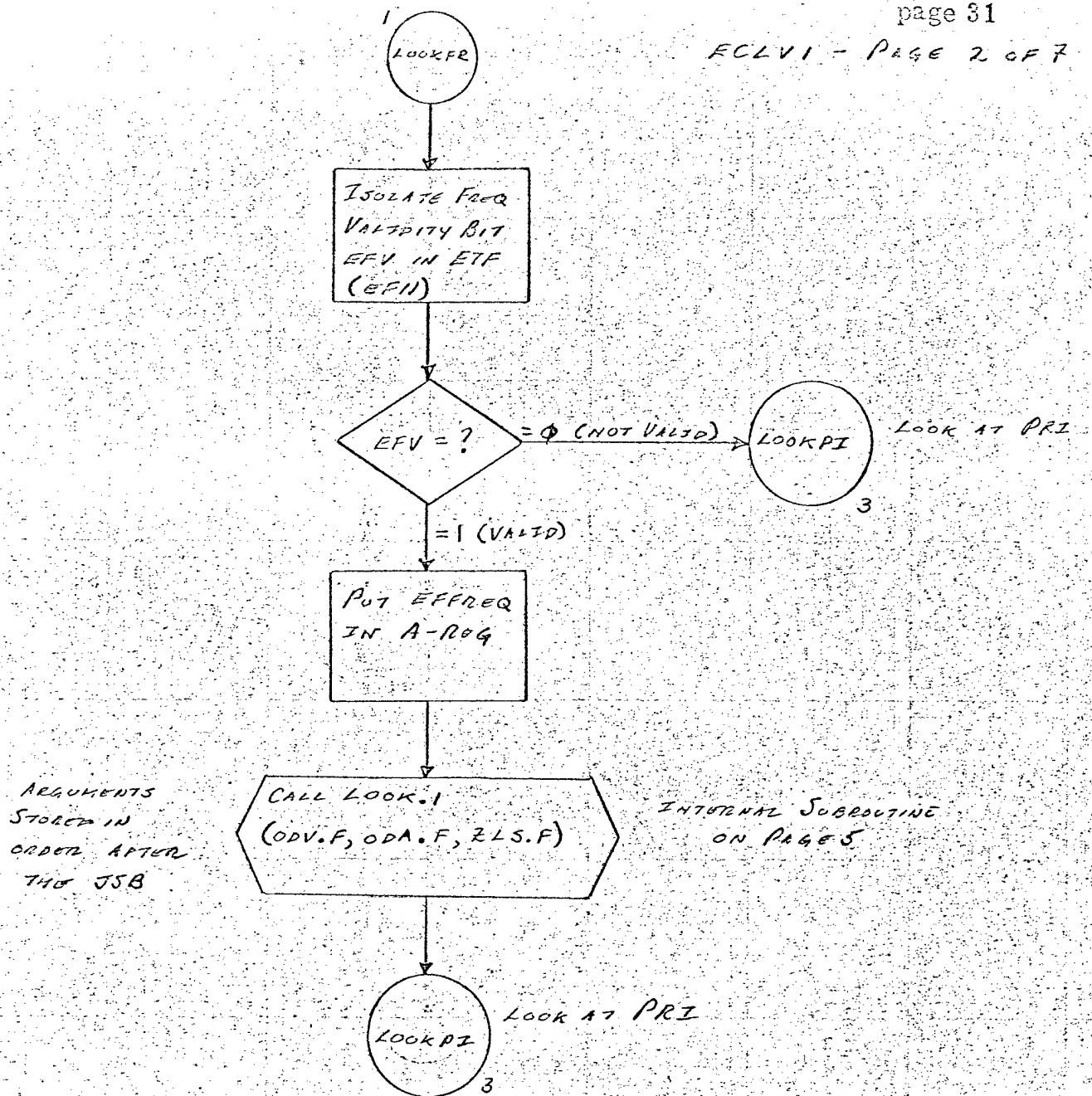
3.2.3.1 ECSTC (page 46)



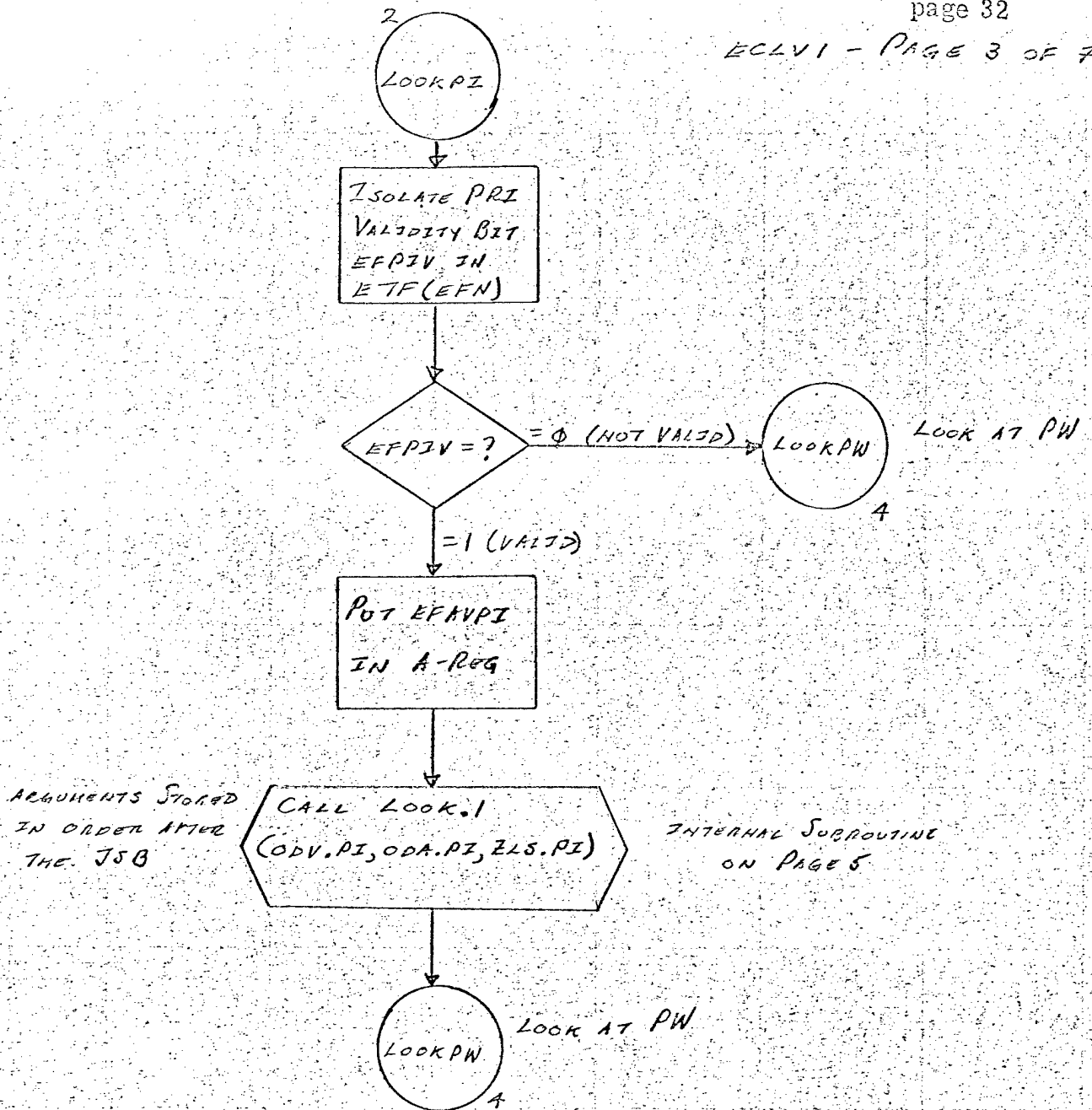
3.2.2 ECLVI

ECLVI - Page 1 of 7

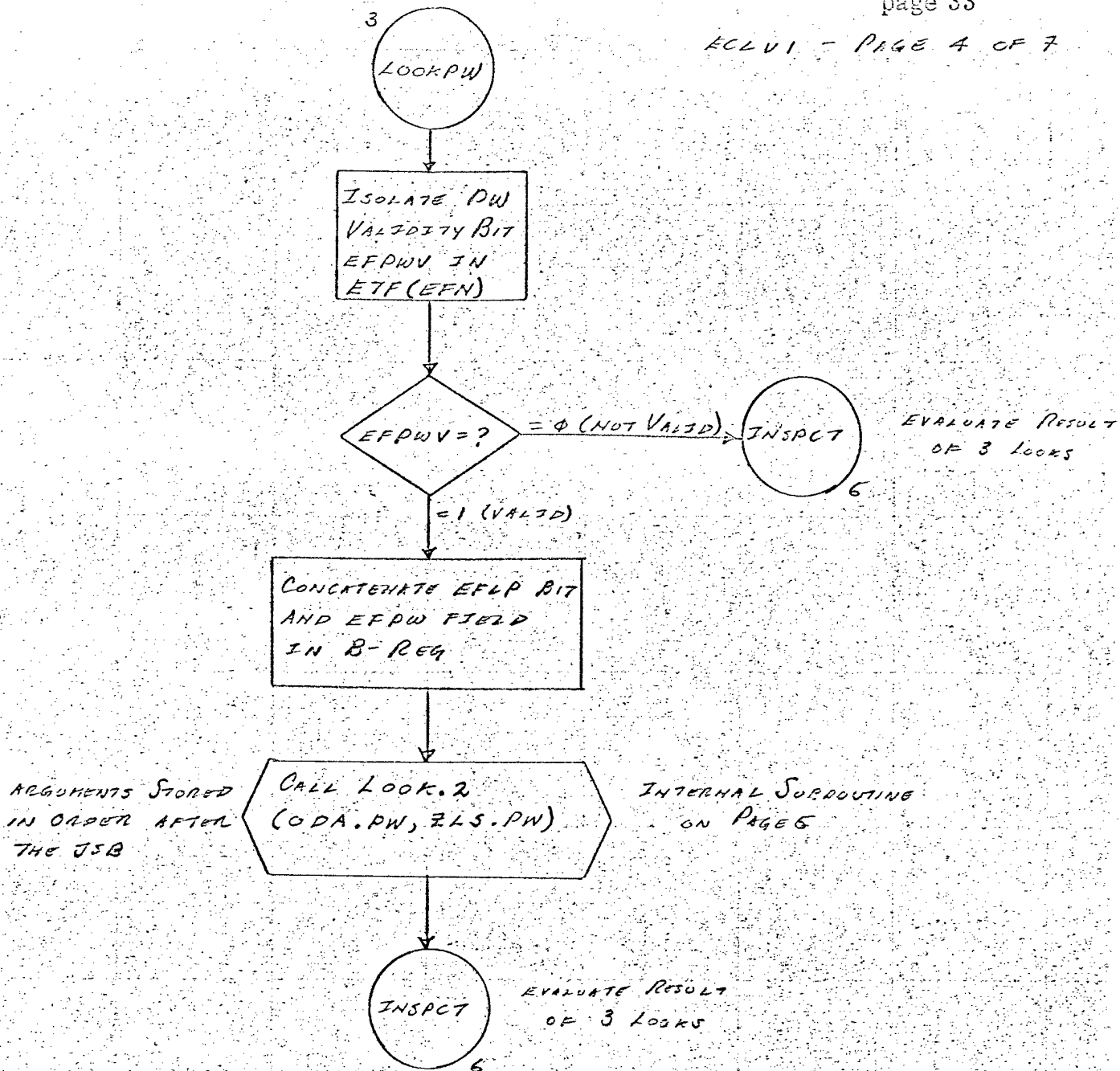




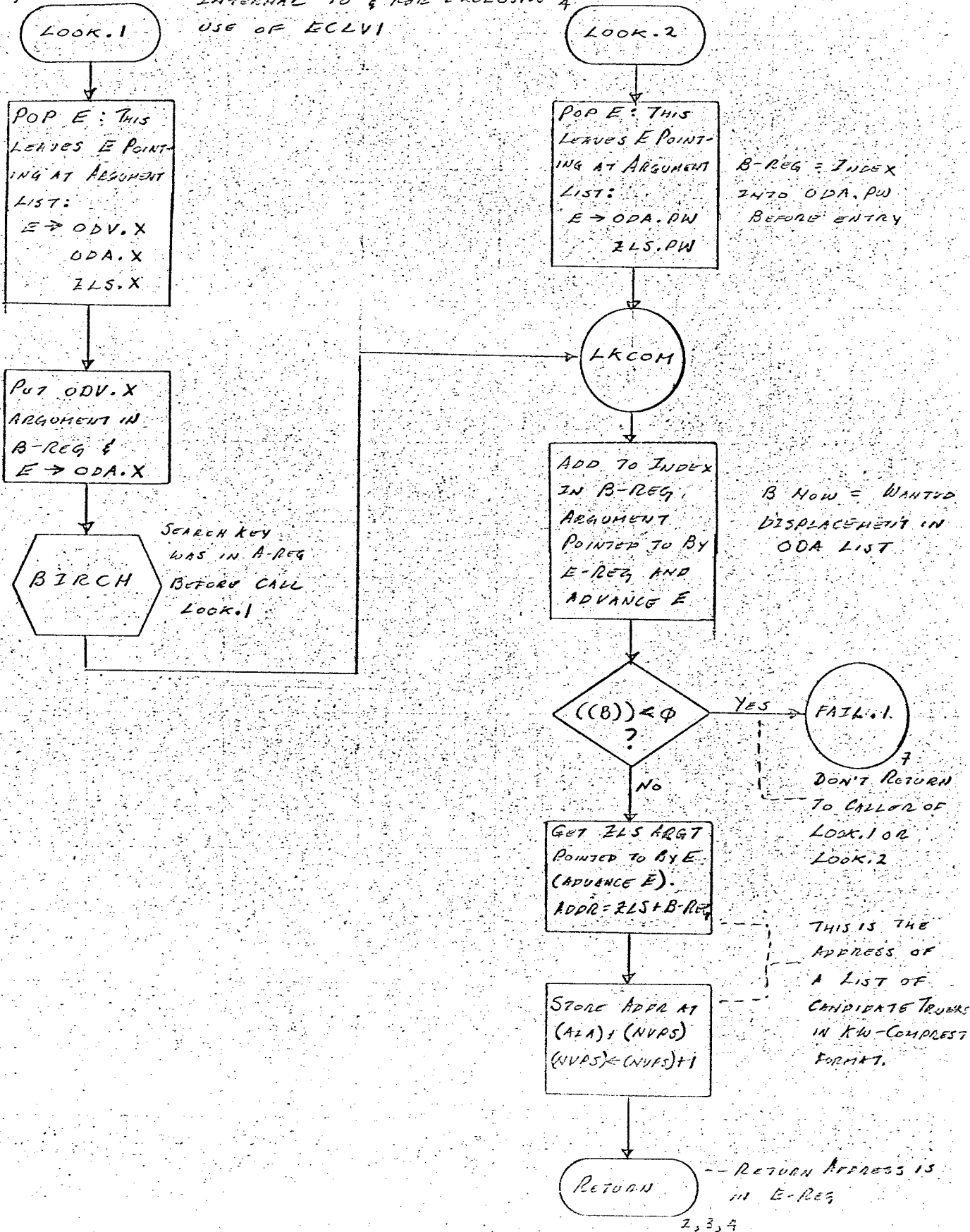
ECLVI - PAGE 3 OF 7

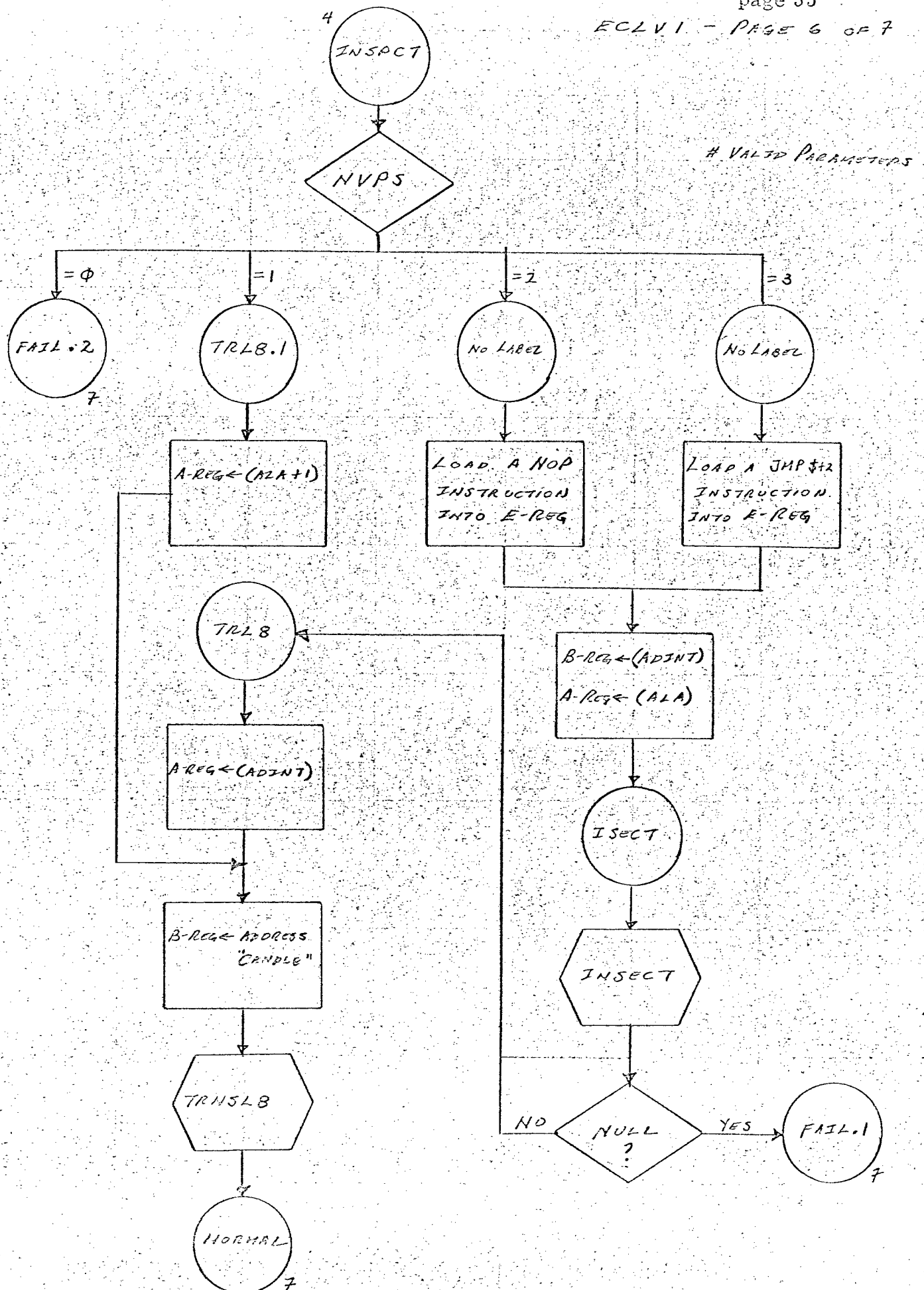


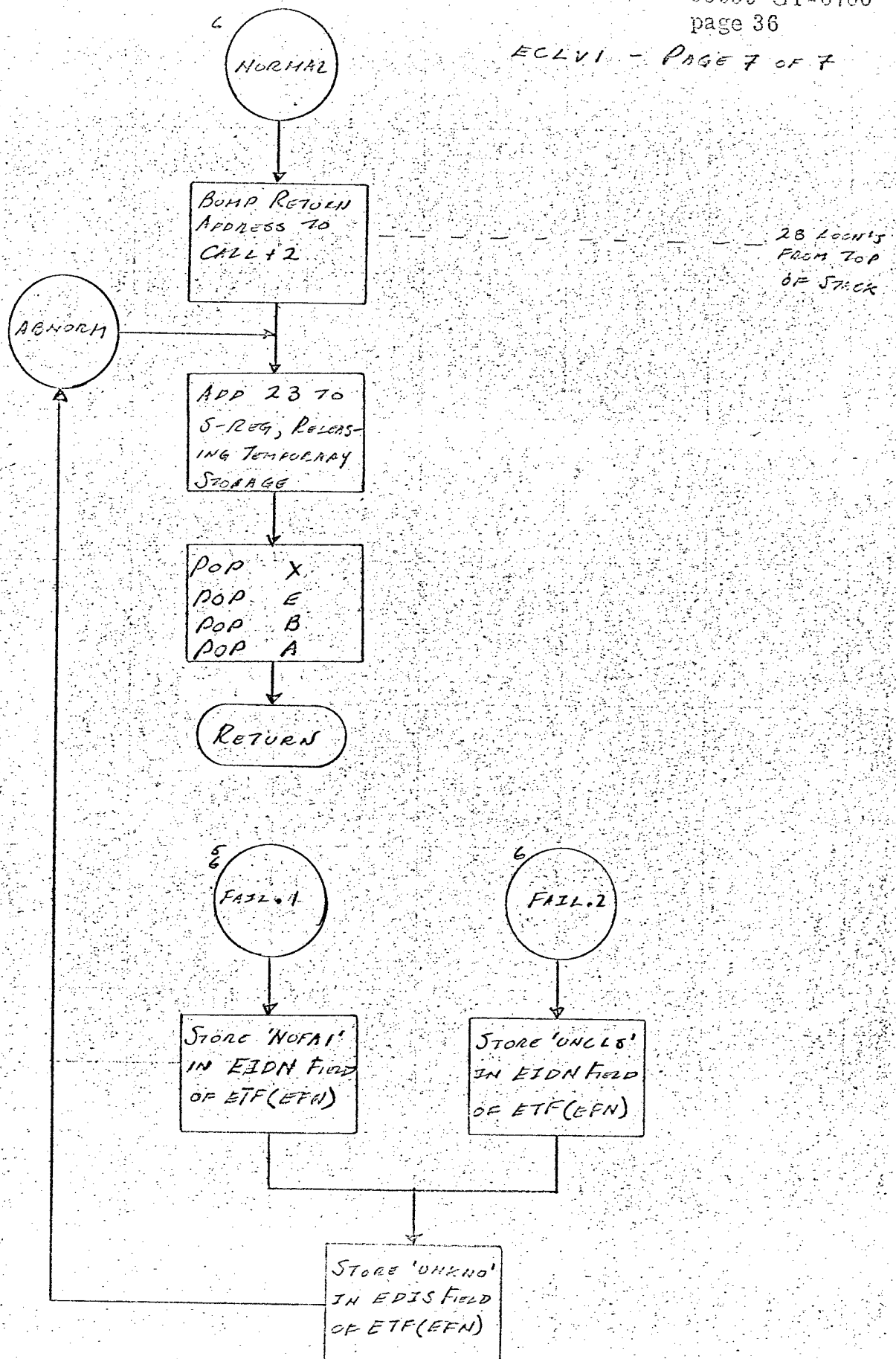
ECLVI - PAGE 4 OF 7



2,3

INTERNAL TO & FOR EXCLUSIVE &
USE OF ECLV1





3.2.2.1 BIRCH

BIRCH - PAGE 1 OF 1

BIRCH

A-REG = SEARCH KEY (SK)

B-REG →

SAVE A-REG
X-REG
E-REG
ON STACK

N
SL₁
SL₂
⋮
⋮
⋮
SL_N

SEARCH LIST
(ASCENDING ORDER)

X ← -1
E ← -N
A ← CI(N)
(INITIAL H)

X IS SHIFT
COUNT; POSITIVE
CHANGE THROUGH
OUT.
CI = 1'S COMPL.

SAVE ADDR
SL₁ ON STACK

B → SL₁

1ST TIME

LOOPG

← WHEN SK > SL₂

LOOPL

← WHEN SK < SL₂

E ← -(M+1)/2

SAD; X X

E ← (M+1)/2

SAD; X X
NEG E, E

COMMON

B ← B - E
E ← A = CI(H)

B = ADDR OF SL₂

E ← CI(H/2)
= CI(NEW H)

SAD; X X

A ← E
A ← E
E ← -M

SK vs SL₂

A ← E
E ← -M

LoopL

H = 0?

B ← B - 1

B ← B - ADDR SL₁
(ADDED)

YES

M = 0

NO

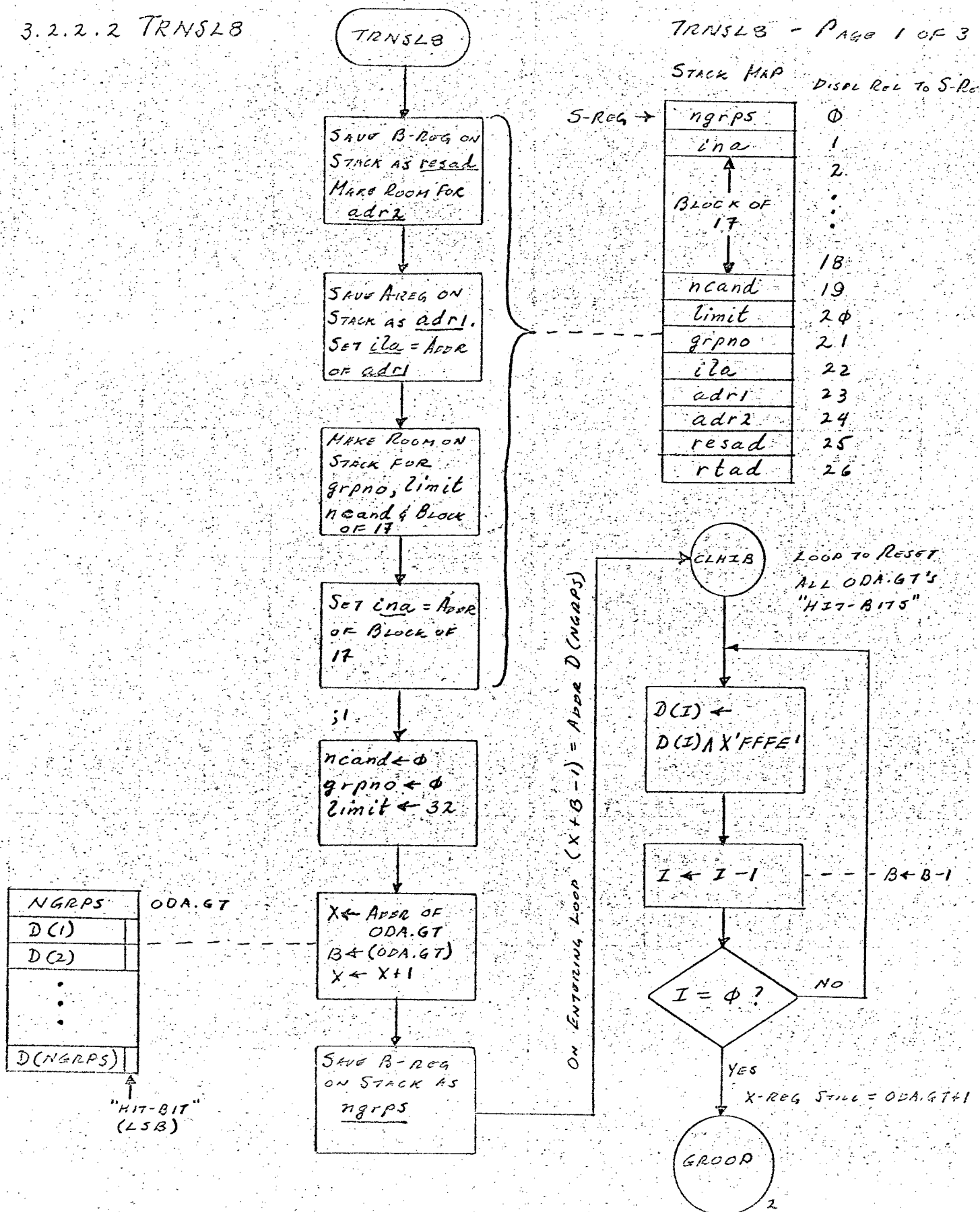
LoopG

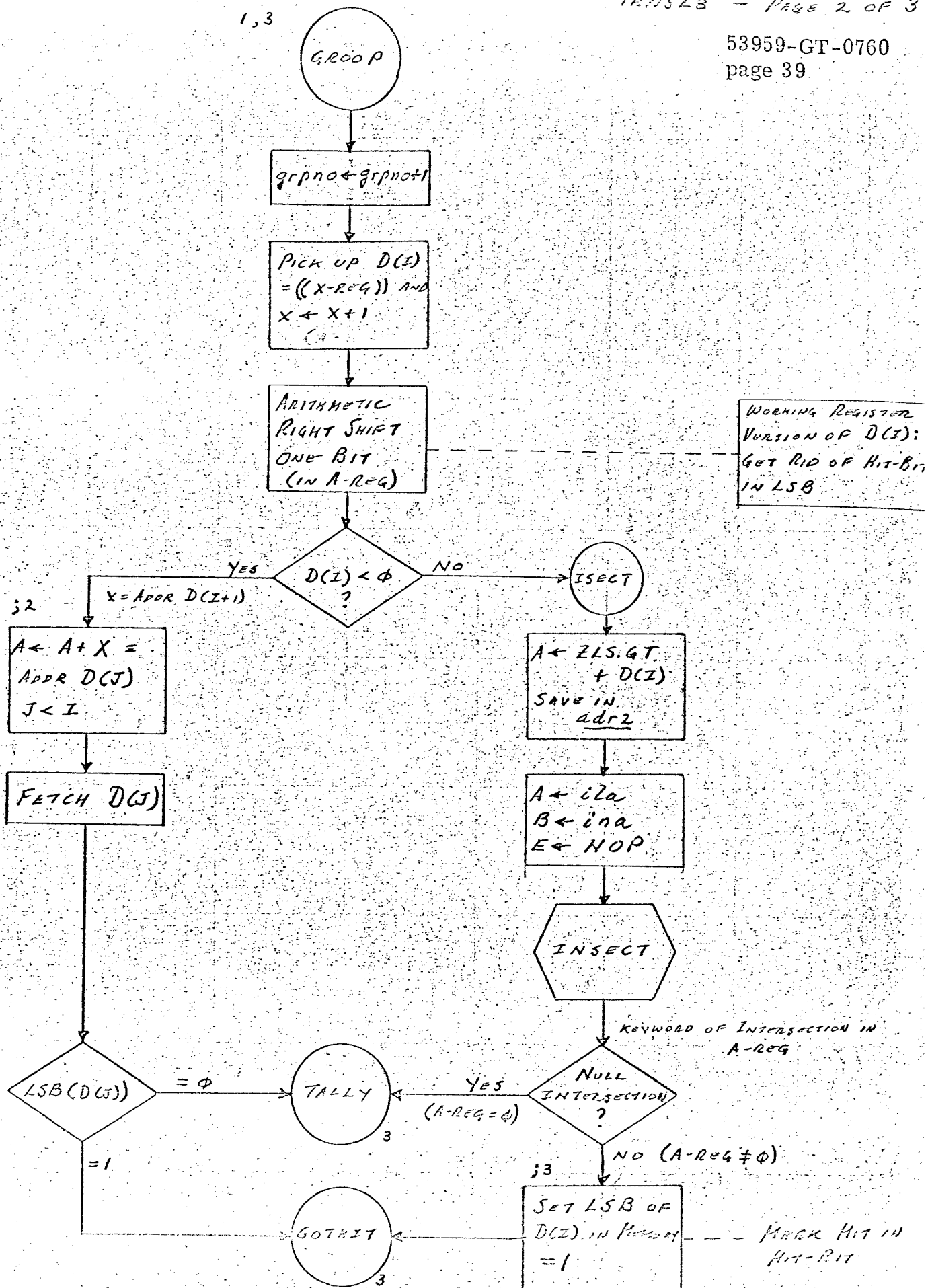
RESTORE E
X
A
REG'S FROM
STACK

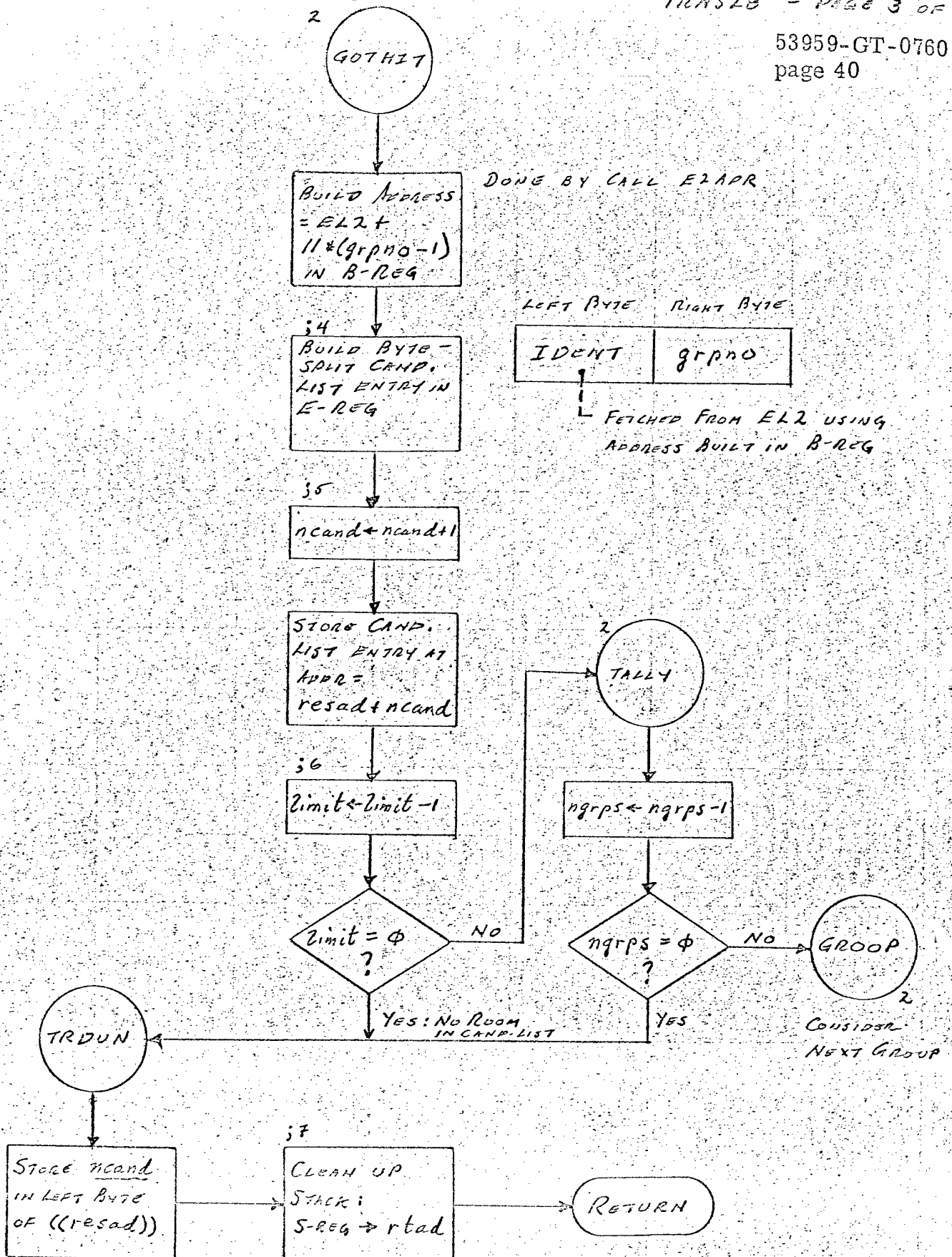
RETURN

3.2.2.2 TRANSLB

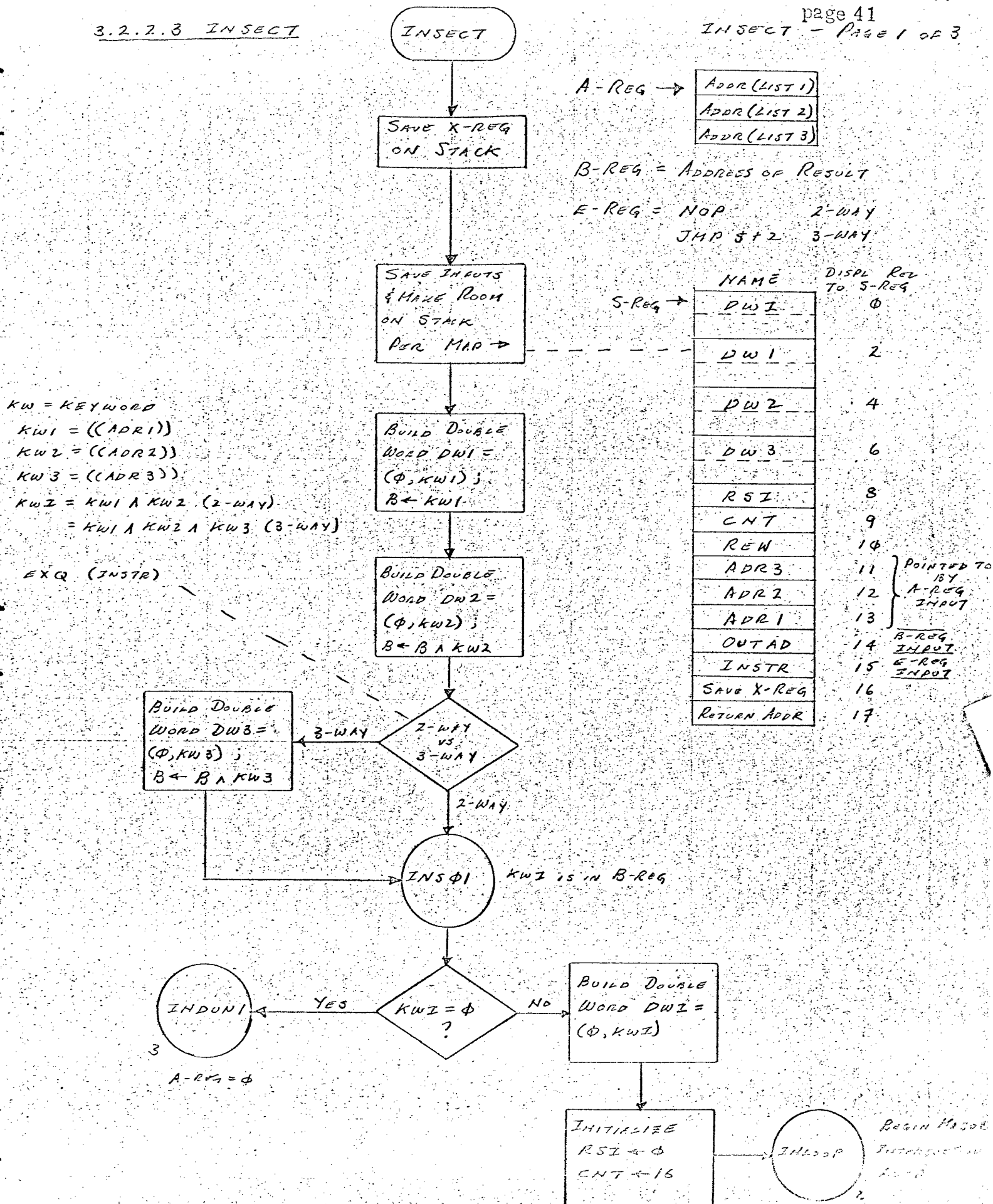
TRANSLB - Page 1 of 3

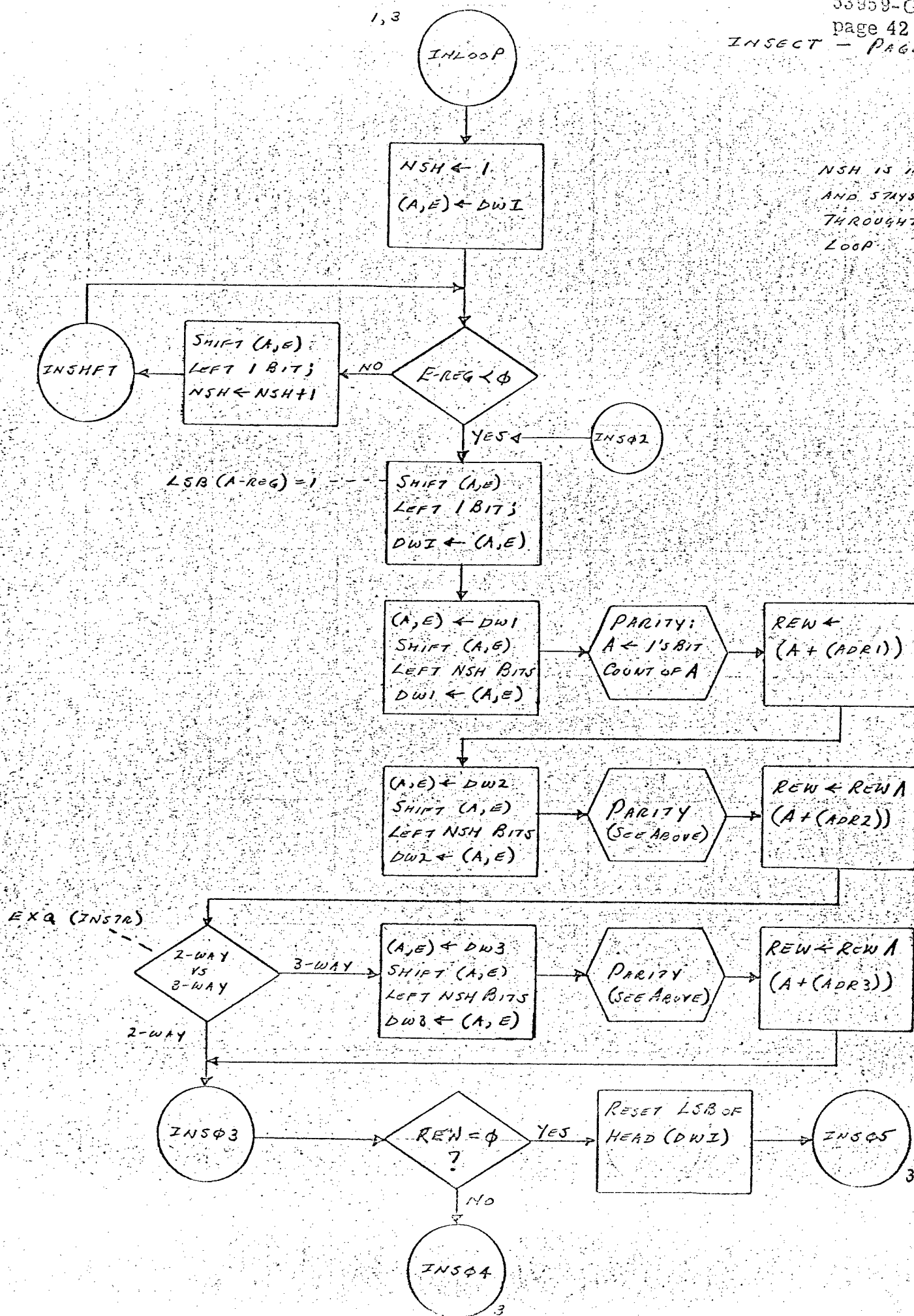


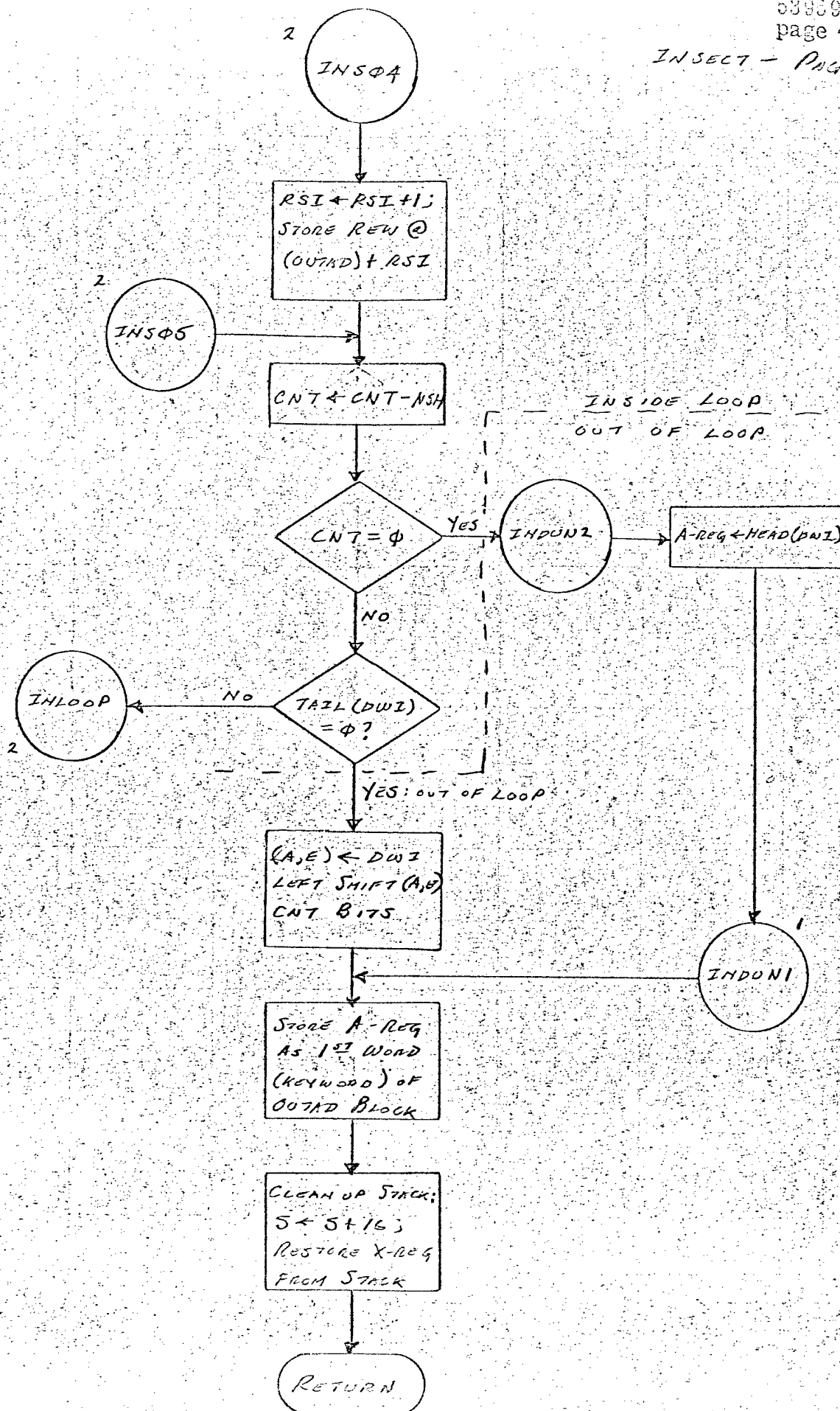


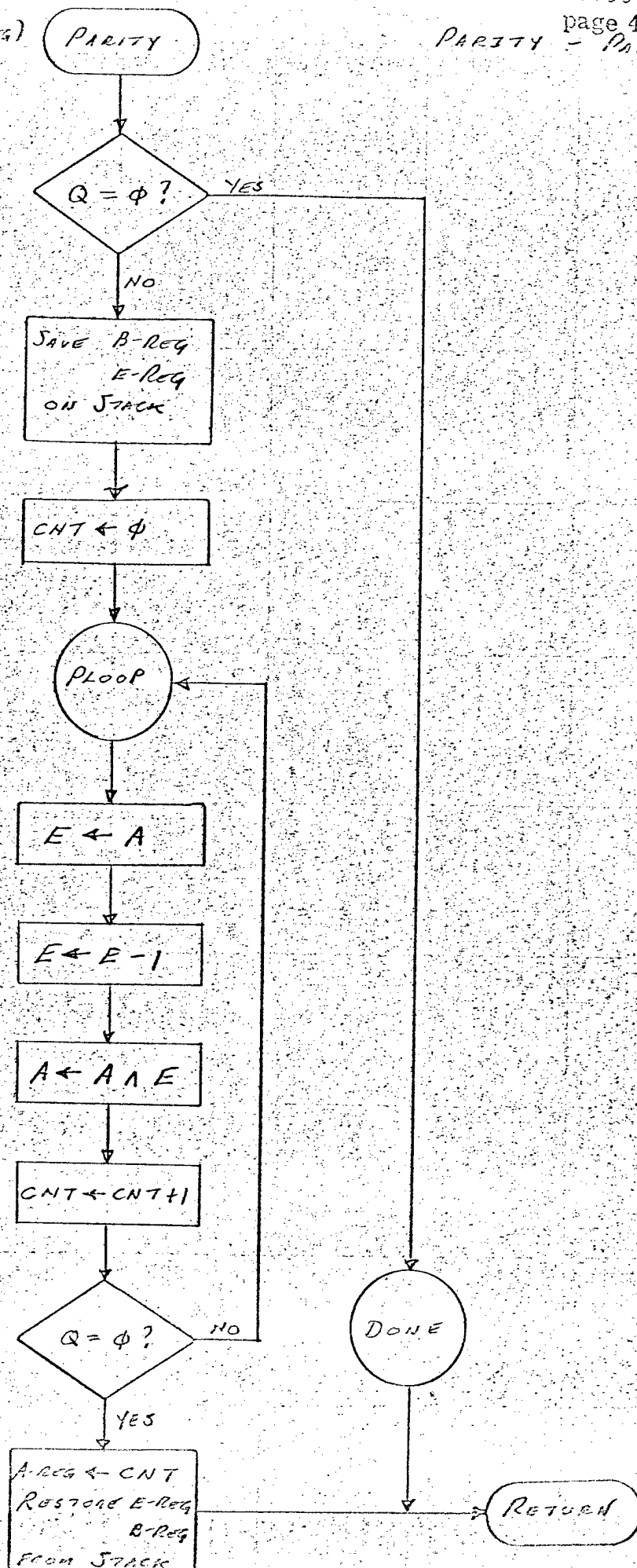


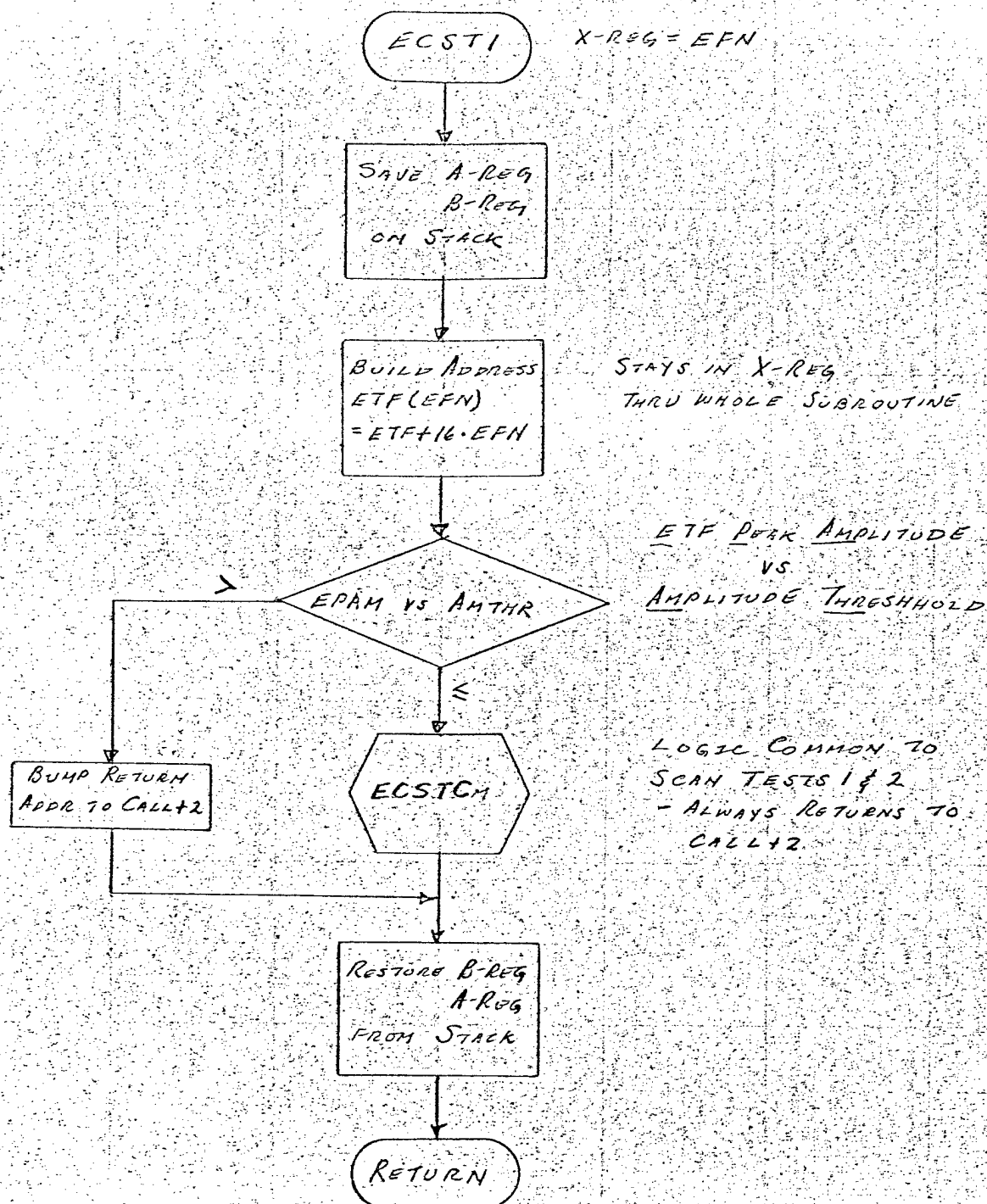
3.2.2.3 INSECT

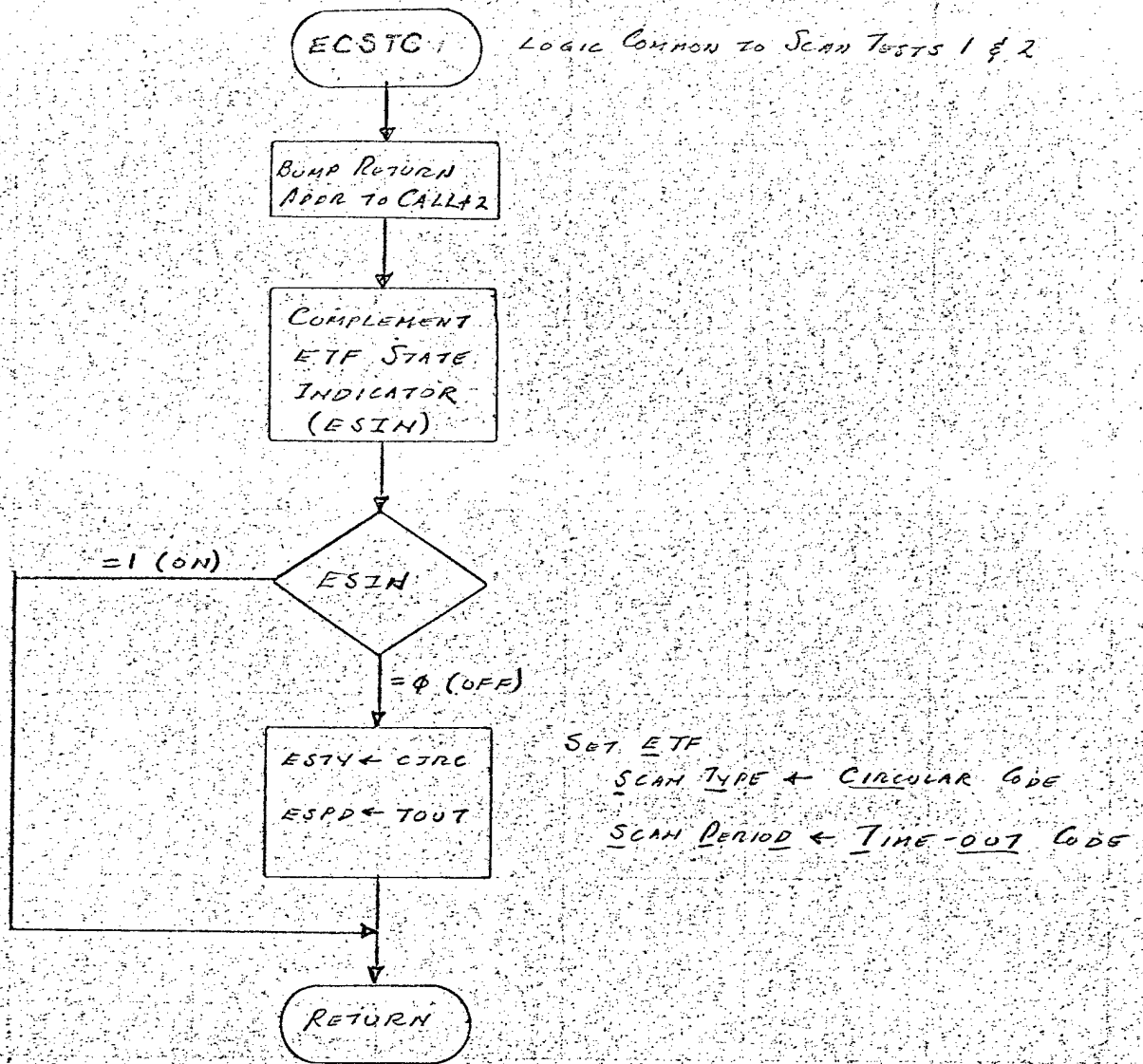






$A-REG \leftarrow \text{BIT COUNT}(A-REG)$ DEFINE ARGUMENT $\equiv Q$ CNT IS KEPT
IN B-REG $Q \leftarrow Q \cdot \text{AND} (Q-1)$ 





3.3 COMPUTER SUBPROGRAM ENVIRONMENT

The Data Environment of this subprogram consists of:

- (a) Global Tables - These are defined in the CDBDD. The specific tables used herein are -

ETF - - Emitter Track File

EL2 - - Emitter Library 2

Local parameters are defined for the displacement and Bit layout templates of these tables. These names are prescribed by the CDBDD. See Section 3.3.3.

- (b) Global Parameters - Values of certain codes. These include:

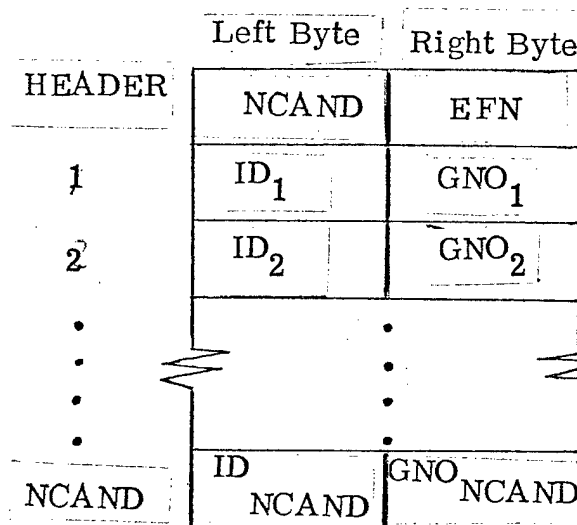
Name	Subroutine	Function
EMNSRQ	ECDR	Executive Message Number of Scan-Analysis-Request
RMCEC1	ECDR	Return Module Code of Emitter Classification Processing - 1
AMTHR	ECST1	Amplitude Threshold
CIRC	SCTCOM	Circular Scan-Type Code
TOUT	SCTCOM	Time-Out Scan Period Code
NOFA1	ECLV1	"None-of-Above" Identity Code
UNCLS	ECLV1	"Unclassified" Identity Code
UNKNO	ECLV1	"Unknown" Display Code

- (c) Local Permanent Storage - - These are tables with fixed addresses. They are described in Section 3.3.1.
- (d) Local Ephemeral Storage - - These are local tables, variables, indices, etc., which are assigned space on the stack upon entry to the subroutines wherein they are needed and disappear upon return therefrom. In our estimation, this factor outweighs their mixed nature. Therefore, we discuss the stack storage for each subroutine as a unit in Section 3.3.2.

3.3.1 Local Permanent Tables

3.3.1.1 CANDLE

- (a) This table is named "CANDLE". It is defined in and known only to subroutine ECLV1.
- (b) CANDLE is used to store the final, Level 1 Candidate List. As such, its address is sent to subroutine TRNSL8 which loads CANDLE with this List. Its address is also returned by ECLV1 to the driver ECDR which stores this address in the scan-analysis request message sent to the Executive.
- (c) CANDLE has 33 locations allotted to it. The first word is a header and the table proper consists of the following 32 words which are indexed by addition of an index NCAND = 1, 2, . . . , 32 to the header's address.
- (d) The structure and bit layout are shown in the accompanying diagram.



NCAND ≤ 32: Number of Candidates

EFN: Emitter File Number (Input)

ID_i: Identity Code of ith Candidate

GNO_i: Group Number of ith Candidate

ID_i is taken from the GNO_ith entry of EL2.

If no Candidate List is generated, the header = EFN,
i.e., NCAND = ∅

3.3.1.2 Emitter Library 1 (EL1)

EL1 consists of data structures whose purpose is to enable the generation of a Candidate List each of whose members (trunks) could match the input Emitter Track File (ETF) on the three parameters frequency (f), pulse repetition interval (π) and pulse width (p).

Accordingly, EL1 consists of three major divisions, one for each physical parameter. Each of these divisions consists of:

- an Outer Directory. Therefore, we have OD. F, OD. PI and OD. PW
- a Zone-List-Section. Therefore, we have ZLS. F, ZLS. PI and ZLS. PW.

We economize on the requisite description by noting that

OD. F and OD. PI are, except for length and physical significance, of identical structure. Therefore, we describe them as OD. X where X stands for either F or PI. Moreover, all three ZLSs are of identical structure, so we describe them as ZLS. Y where Z stands for either F, PI or PW.

3.3.1.2.1 Outer Directories

3.3.1.2.1.1 OD. X: Outer Directories OD. F and OD. PI

- (a) The name of this locally defined and used table is "OD. F" ("OD. PI"). It consists of two sections: a value section "ODV. F" ("ODV. PI"), and an address section "ODA. F" ("ODA. PI").
- (b) ODV. F (ODV. PI) contains N. F (N. PI) values of frequency (PRI) in ascending order. The first value is much less than any expected value of frequency (PRI) and the last much greater. The table exists from the beginning of the program (not created dynamically), and is used as input to the binary search subroutine (BIRCH). The result of this search is an index L such that the input value of frequency, f (PRI, π) is

$$\begin{aligned} \text{ODV. F (L)} &\leq f < \text{ODV. F (L + 1)} \\ [\text{ODV. PI (L)} &\leq \pi < \text{ODV. PI (L + 1)}] \end{aligned}$$

3.3.1.2.1.1

(b) -continued -

ODA. F (ODA. PI) contains N. F (N. PI) pointers. If L is as just described and $ODA. F (L) < \emptyset$ ($ODA. PI (L) < \emptyset$) then there are no known emitters with the frequency f (PRI π); whereas, if $ODA. F (L) \geq \emptyset$ ($ODA. PI (L) \geq \emptyset$) then this pointer is in displacement D (L) such that the address

$$A (L) = ZLS. F + D (L)$$

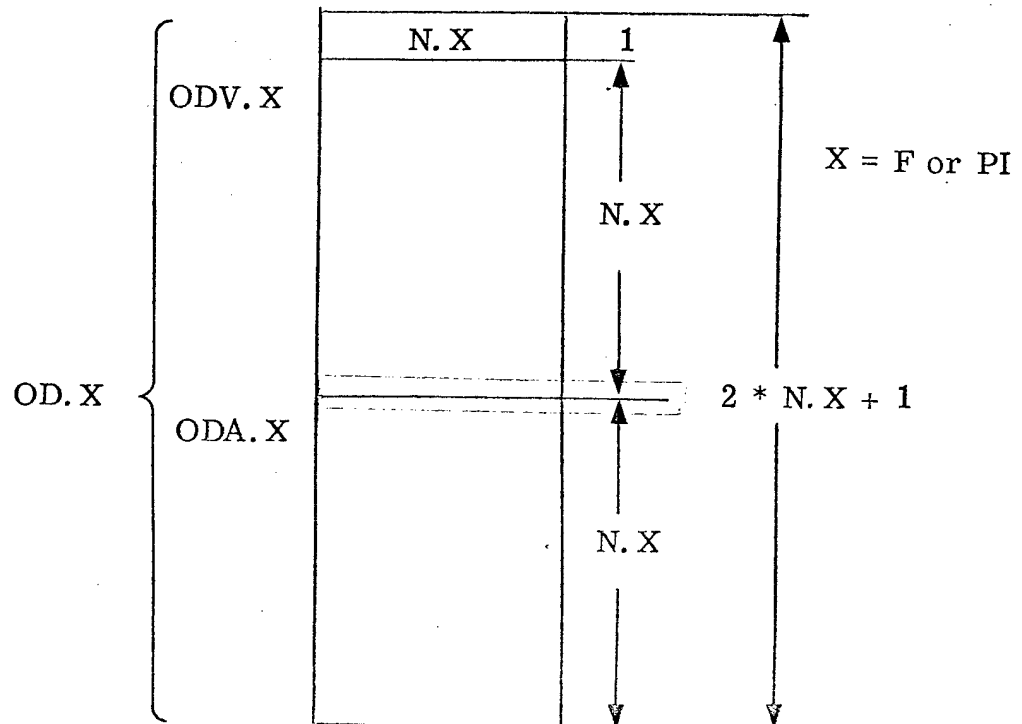
$$[A (L) + ZLS. PI + D (L)]$$

points to a list of emitters in ZLS. F (ZLS. PI) which are exhibit the given value of f (π).

(c) The length of OD. F is $2 * N. F + 1$.
(OD. PI) $(2 * N. PI + 1)$

Indexing is applied to the two subsections ODV. F and ODA. F (ODV. PI and ODA. PI) by adding the index of the desired entry to the addresses ODV. F and ODA. F (ODV. PI and ODA. PI) where the index runs from \emptyset to N. F - 1 (N. PI - 1).

(d) OD. F (OD. PI) is structured as depicted below. Bit layout is not applicable; each item is a whole word datum.



3.3.1.2.2 Zone List Sections

- (a) The name of these tables are "ZLS.Y" where Y = F, PI or PW.
- (b) Each such table contains more than 1 but no more than N.Y subsections ($N.PW \equiv 32$). Each subsection is an enclosed list of the emitter trunks which can exhibit the value of the corresponding parameter as described in Section 3.3.1.2.1.1(b) and 3.3.1.2.1.2(b). Each such table exists from the beginning of the program (not created dynamically).
- (c) The size of ZLS.Y depends on the number of non-negative entries in ODA.Y. This number is of no importance to the execution of the ECLV1 programs and so is not assigned a symbolic name. However, for the sake of this sizing discussion alone we define

$$\begin{aligned} \mathcal{S}.Y(I) &= \emptyset \text{ if } ODA.Y(I) < \emptyset \\ &= 1 \text{ if } ODA.Y(I) \geq \emptyset \end{aligned}$$

$$\begin{aligned} L.Y(I) &= \text{if } ODA.Y(I) < \emptyset \\ &\in [2, 17] \text{ if } ODA.Y(I) \geq \emptyset \end{aligned}$$

$$\text{Then the size of ZLS.Y} \leq \sum_{c=1}^{N.Y} \mathcal{S}.Y(I) * L.Y(I)^{(1)}$$

$$\text{ZLS.Y consists of } \sum_{c=1}^{N.Y} \mathcal{S}.Y(I)^{(1)} \text{ subsections}$$

each of variable length (see d below) ranging from 2 to 17 words. Indexing is accomplished in two stages. First, locate the subsection initial address A(Y) as described in the above cited sections. Then use the bit-encoding as described below.

- (d) Each subsection of ZLS.Y consists of a leading (key) word and one or more non-zero binary vector words. ⁽²⁾

Each subsection contains an emitter trunk presence (1)/absence (\emptyset) bit-vector where the number of trunks

⁽¹⁾ Inequality applies because identical subsections need be stored only once.

⁽²⁾ This format is referred to keyword-compressed/binary-vector format or KWC/BV format elsewhere in this document and in the commentary of the program listings.

RAYTHEONRAYTHEON COMPANY
LEXINGTON, MASS. 02173

CODE IDENT NO.

49956

SPEC NO.

53959-GT-0760

SHEET

53 OF

REV

3.3.1.2.2

(d) -continued-

$NTRUNX \leq 256^{(3)}$. Accordingly, a subsection covering all 256 trunks would be 16 words long with MSB (word 1) corresponding to trunk #1 and LSB (word 16) corresponding to trunk #256. Since, in actuality, most bits and most words will be = \emptyset , space is saved by using the keyword as a map of the bit-vector words that are non-zero (Have ≥ 1 bits = 1) and omitting all pure zero words.

Thus, by way of example, consider a subsection which is to include trunks 1, 3, 18, 80 and 113. The bit vector before the keyword compression would be as shown in Figure 1a. The subsection after the keyword compression as actually stored is shown in Figure 1b.

(3)

 $NTRUNX = N.F$

RAYTHEON

RAYTHEON COMPANY
LEXINGTON, MASS. 02173

CODE IDENT NO.

49956

SPEC NO.

53959-GT-0760

SHEET

54 OF 89

REV

Word #																	Trunk #'s
1	1	∅	1	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	1, 3
2	∅	1	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	18
3	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	
4	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	
5	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	1	8∅
6	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	
7	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	
8	1	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	113
9	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	
10																	
11																	
12																	
13																	
14																	
15																	
16	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	

Figure 1a. Bit Map of Trunks 1, 3, 18, 8∅, 113
Before Keyword Compression

Word #																	Keyword: Words
1	1	1	∅	∅	1	∅	∅	1	∅	∅	∅	∅	∅	∅	∅	∅	1, 2, 5, 8 ≠ ∅
2 (1)	1	∅	1	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	
3 (2)	∅	1	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	
4 (5)	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	1	
5 (8)	1	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	

Figure 1b. Bit Map of Trunks 1, 3, 18, 8∅, 113
After Keyword Compression

3.3.1.3 Library-to-Library Linkage Tables

Library EL1 (Local to the subroutines of this document) is based on emitter trunks which are characterized by unique sextuplets of the physical measurements

Frequency - minimum and maximum

PRI - minimum and maximum

PW - minimum and maximum

arranged and numbered on ascending minimum frequency.

Library EL2 is defined in the CDBDD. It is based on emitter groups which are derived from classification according to scan to type, minimum and maximum scan period, EW technique and related factors.

The two tables of this section are used to associated groups with trunks. The first such table is an outer directory ODA.GT (GT = Group/Trunk), and the second, ZLS.GT, is entirely analogous to the ZLS tables described in Section 3.3.1.2.

3.3.1.3.1 ODA.GT

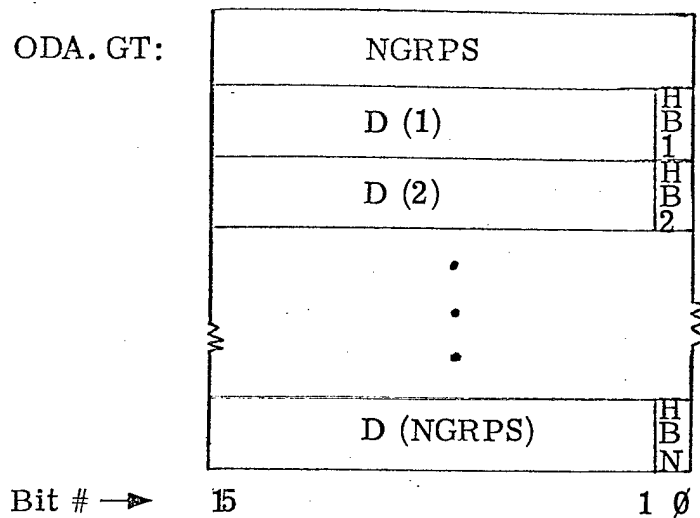
- (a) The name of this table is ODA.GT.
- (b) An element in ODA.GT at address ODA.GT. + I, $I \geq 1$ will be referred to as D(I). Each $D(I) \geq \emptyset$ is a displacement which when added to the address ZLS.GT yields the address of a KWC/BV trunk list in ZLS.GT whose members are associated with the I^{th} group.

A value of $D(I) < \emptyset$ is taken to be $= J - (I + 1)$ for some $J < I$. This indicates that the I^{th} group has the same KWC/BV trunk list in ZLS.GT as the J^{th} group.

Each element D(I) whether \geq or $< \emptyset$ is stored as twice the nominal value discussed above, so that upon being fetched it must be arithmetically shifted right one bit position. This allows the LSB of each (D(J)) to be used to hold a Hit-Bit (flag). See Section 3.1.2.2.

3.3.1.3.1 -continued-

- (c) ODA.GT is NGRPS in length. NGRPS is stored at location ODA.GT itself (see below). Indexing is accomplished by adding the desired index I , $1 \leq I \leq \text{NGRPS}$ to address ODA.GT.
- (d) ODA.GT structure and bit layout are indicated on the accompanying diagram.



ODA.GT: Structure & Bit Layout

3.3.1.3.2 ZLS.GT: - Group Lists of Trunks

- (a) The name of this table is ZLS.GT. The name has been chosen because of its exact identity in use and structure to the ZLS tables of EL1 (Section 3.3.1.2). It has nothing to do with measurement zones of physical parameters.

The other relevant descriptions of ZLS.GT can be gotten from Section 3.3.1.2.

3.3.1.4 Scan Analysis Request Message

- (a) The name of this table is SRQMSG
- (b) This table contains the codes and parameters required to cause the Executive to schedule a scan analysis. It is defined in and known only to the driver ECDR which passes its address to the Executive when it calls EXMES.
- (c) This table is five long; only three of its elements are filled by ECDR. These are accessed by individual labels assigned to the three target elements.
- (d) The structure, bit layout and labels of SRQMSG are shown below.

<u>Label</u>		<u>Contents</u>
SRQMSG:	<u>EMNSRQ</u>	Executive Message # of Scan Analysis Request
	3	Number of Words to Follow
SRRMCD:	<u>RMCEC 2</u> <u>EFN</u>	<div> <div>Left Byte</div> <div>Return module</div> <div>code to EC2</div> </div> <div> <div>Right Byte</div> <div>Emitter file # -</div> <div>Input to ECDR</div> </div>
SRCLAD:	<u>CANDLE</u>	Address of Candidate List
SRREQW:	<div> <div>A</div> <div>W</div> <div>- dc -</div> <div>1</div> <div>-</div> </div>	<div> <div>AW = { 0 }</div> <div>as analysis { is</div> <div>{ 1 }</div> <div>{ is not }</div> <div>wanted</div> </div> <div> <div>Bit 0 = 1</div> <div>Scan analysis</div> </div>
	15 0	

Items filled in by ECDR are underlined.

3.3.2 Local Ephemeral Storage

Three of the subroutines of this subprogram use the stack for their local tables and variables during execution, releasing the storage upon exit. The other four subroutines and the driver (ECCR) use only small amounts of stack storage to save and restore registers. This latter use of the stack will not be described.

An oversimplified example of stack usage for ephemeral storage can be gained from the following example.

A subroutine preserves all registers and needs room for 3 variables named "AL", "EE", and "OOP" and a table 5 long named "THING".

1) Define the stack map with parameters

AL =	0	} Local Ephemeral Storage
EE =	1	
OOP =	2	
THING =	3	
XSAV =	8	} Register Save Area
ESAV =	9	
BSAV =	10	
ASAV =	11	
RTAD =	12	Return Address

2) On Entry do:

PSH	A
PSH	B
PSH	E
PSH	X
SUB, S	P, INC
	XSAV

3) The stack memory map is

	Contents	Displacement Rel to S-Reg
S-Reg →	al	AL = 0
	ee	EE = 1
	oop	OOP = 2
	thing	THING = 3
		4
		5
		6
		7
	Save X-Reg	XSAV = 8
	Save E-Reg	ESAV = 9
	Save B-Reg	BSAV = 10
	Save A-Reg	ASAV = 11
	Return Addr	RTAD = 12

4) In the body of the program, stack items, including the saved registers and the return address can be accessed symbolically and mnemonically. For example

- oop ← oop + 1

Is coded ISZ OOP, S

- al ← al + ee

Is coded LDS AL, S

ADD EE, S

STS AL, S etc.

5) Upon leaving do: ADD, S P, INC
X SAV

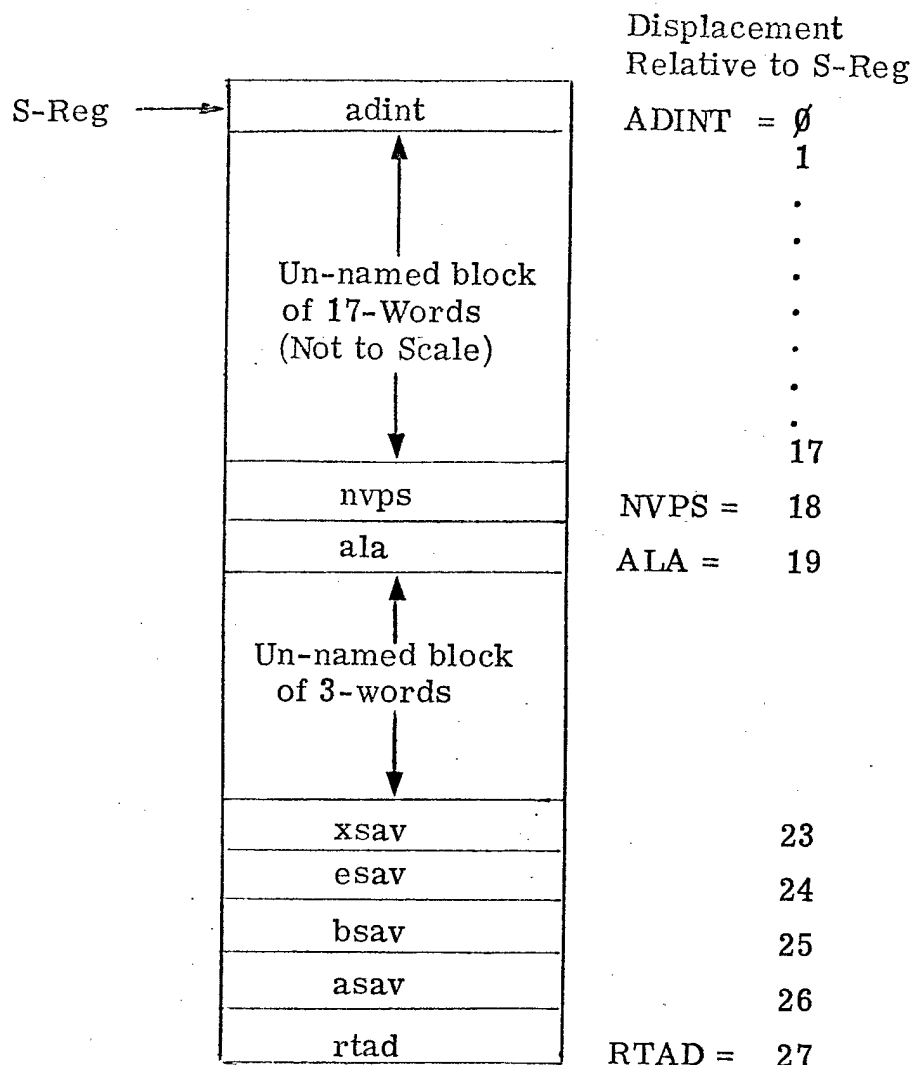
POP X

POP E

POP B

POP A

EXIT



3.3.2.1 Stack Map for ECLV1

adint is also loaded into the A-Reg before calling TRNSL8. This tells TRNSL8 where to find the list it is to translate.

un-named } As implied by the description of adint, this holds
block of } the KWC/BV format intersection output of INSECT
17-words } which is also input to TRNSL8.

nvps = Number of valid parameters = 1's bit count of the three validity bits

EFV
EPIV
EPWV

It is set = \emptyset at start of ECLV1 and augmented each time one of the validity bits is inspected and found non-zero.

ala = Address of list of addresses. This is initialized at start of ECLV1 to = the absolute address of the following location (1st word of block of 3 words), and does not change throughout the execution of ECLV1.

After a validity bit has been found $\neq \emptyset$ and after the index into the ODA list has been found by binary search or direct lookup, but before nvps has been augmented, ala + nvps is used as the store address for the address.

ODA element + ZLS base address

ala is loaded into the A-Reg before calling INSECT. It points to the 2 or 3 input list addresses used by INSECT.

un-named } Use is as described under ala.
block of }
3-words }

xsav - Requires a special comment. This is not the input value (EFN) of X-Reg, but = address of table CANDLE. Restoring X from stack on leaving sets X as required.

3.3.2.2 Stack Storage for Subroutine TRNSL8

The stack map is shown in an accompanying diagram. The table and item descriptions follow.

ina = Intersection address. Exactly analogous to adint described above as regards contents and use in calling INSECT. It is not, of course, used in calling TRNSL8.

un-named
block of
17-words } INSECT must have a place to store its result. This is it when INSECT is called from TRNSL8. However, the result itself is not wanted or used, only the indication of nullity/non-nullity of the intersection. (Candidate for future modification - - this is silly)

ncand - Number of candidates. Initialized to \emptyset . Each time a candidate group is recognized, it is incremented and then used as an index to create address at which to store candiate entry (resad + ncand).

When all candidates have been found (hit bottom of ODA.GT or run out of room), ncand is stored in the left byte of the result address (resad).

grpno - A running group number index initialized at \emptyset and augmented each time a new entry of ODA.GT is picked up. It is used as an entry index into EL2 by means of the address computation

$$EL2 (grpno) = EL2 + 11 * grpno$$

grpno is also stored in the right byte of the entries of CANDLEas discussed in Section 3.3.1.1.

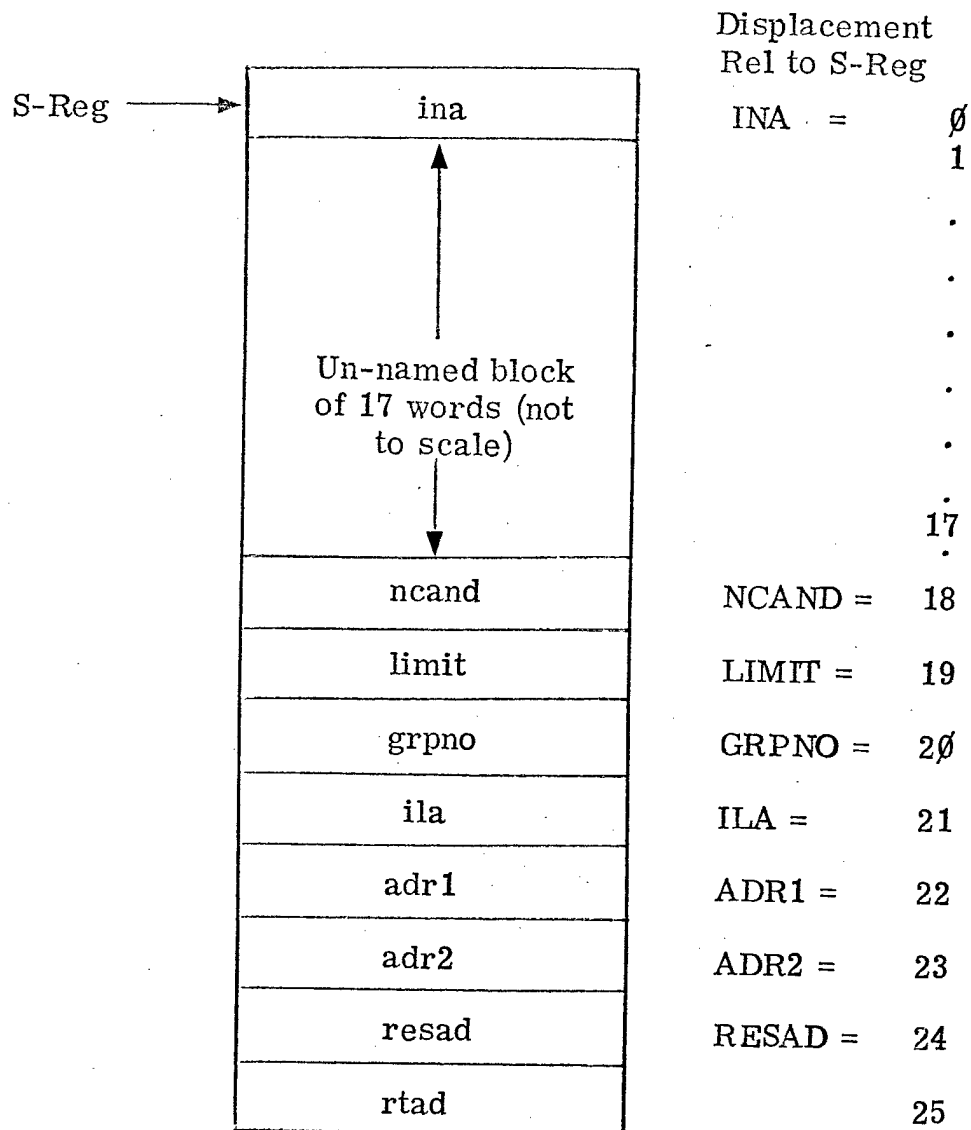
ila - Absolute address of following item (adr1). Exactly analogous to ala above.

adr1 - Place where the address of the KWC/BV format emitter trunk list (input in A-Reg) is stored.

adr2 - Place where address

ODA.GT(I) + ZLS.GT is stored.

resad - Result address. This was input in B-Reg.



3.3.2.2 Stack Map for TRNSL8

3.3.2.3 Stack Storage for Subroutine INSECT

The stack map is shown in an accompanying diagram. The item descriptions follow.

dwi - Double Word Intersection = Initialized to (\emptyset , KWI)
where

$$KWI = KW1 \wedge KW2 \quad \text{or}$$

$$KWI = KW2 \wedge KW3 \quad KW3$$

For 2 and 3-way intersections, respectively and kwi is the keyword of the i^{th} input list.

At the end of the routine, $DWI = (KWI', \emptyset)$ where KWI' is the true keyword of the result.

dw1 = Double Word 1 = (\emptyset , KW1) initially.

dw2 = Double Word 2 = (\emptyset KW2) initially.

dw3 = Double Word 3 = (\emptyset , KW3) initially. If the call on Insect requires a 2-way intersection, the space for dw3 is still allotted but it is not used.

rsi = Result Storage Index. Initially = \emptyset , rsi is incremented each time a non-zero result word (see rew) is found. This result word is stored at address

$$\text{outad} + \text{rsi}, \text{rsi} = 1, 2, \dots, 16.$$

cnt = Count of bits not yet shifted left out of DWI - tail into DWI-head. Initially = 16, cnt is diminished at the end of each iteration by nsh (maintained in X-Reg) the number of bits DWI had to be shifted to get LSB $[\text{Head (DWI)}] = 1$. The condition $\text{cnt} = \emptyset$ is a sufficient, but not necessary, condition for the end of the algorithm.

rew = Result Word. This location is used as a temporary repository of the partial 2 or 3-way intersection of words from the input lists.

RAYTHEONRAYTHEON COMPANY
LEXINGTON, MASS. 02173

CODE IDENT NO.

49956

SPEC NO.

53959-GT-0760

SHEET
66 OF

REV

Displacement
Rel to S-Reg

S-Reg →

dwi
dw1
dw2
dw3
rsi
cnt
rew
adr3
adr2
adr1
outad
instr
xsav
rtad

DWI =	Ø
DW1 =	2
DW2 =	4
DW3 =	6
RSI =	8
CNT =	9
REW =	1Ø
ADR3 =	11
ADR2 =	12
ADR1 =	13
OUTAD =	14
INSTR =	15
	16
	17

3.3.2.3 Stack Map for INSECT

adr3 }
adr2 }
adr1 }

At entry, A-Reg →

ADDR (List 1)
ADDR (List 2)
ADDR (List 3)

These 3 items are transferred to adr1, adr2 and adr3 (even if ADDR (List 3) is fictitious). The inverted order is due to the fact that stack building proceeds in bottom-to-top order.

outad = Output Address, i.e., where to store the intersection result. This was input at entry time in B-Reg. It is used as indicated under rsi for storing binary vector result words. At end of algorithm, KWI is stored at (outad).

instr = Instruction to switch INSECT's logic to 2 or 3-way intersection as required. This is input at entry time in E-Reg.

instr = NOP for a 2-way intersection
= JMP \$ + 2 for a 3-way intersection

The mechanics are shown in the accompanying diagram. instr is made active by use of the RP-16 EXQ (Execute) instruction and is used at two points in the logic:

- a) At point of fetching KW3, storing DW3 and ANDing KW3 into KWI.
- b) At point of fetching binary vector word from third input list and ANDing it into the partial result word (rew).

RAYTHEON

RAYTHEON COMPANY
LEXINGTON, MASS. 02173

CODE IDENT NO.

49956

SPEC NO.

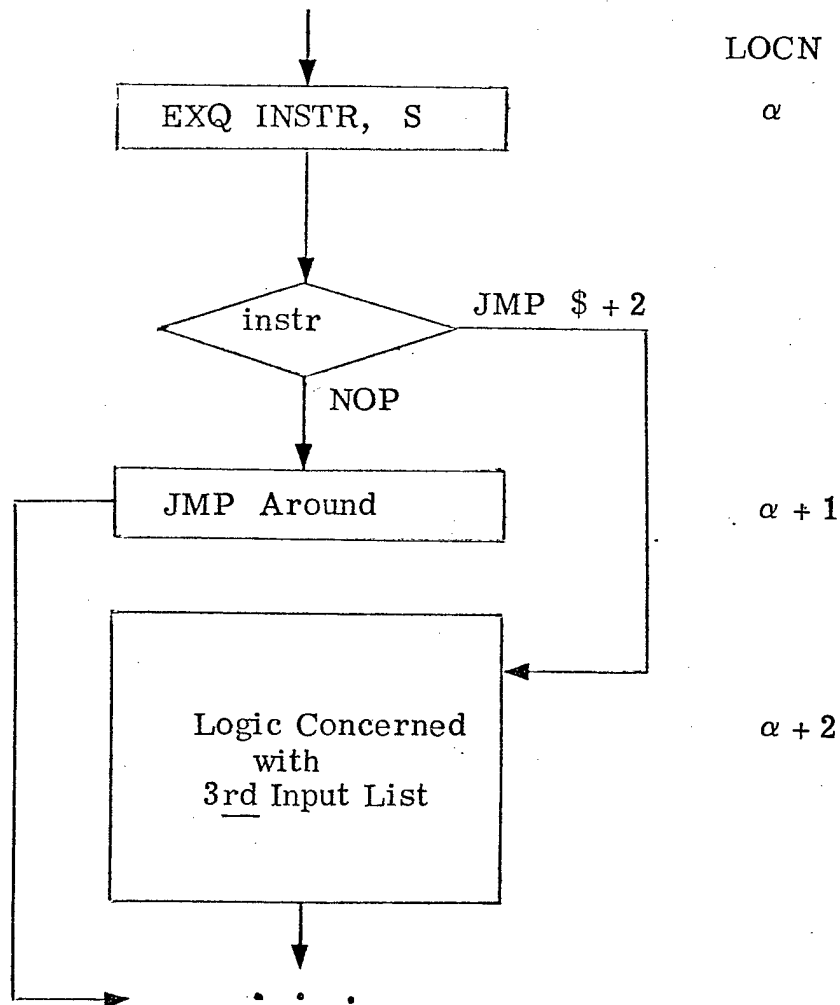
53959-GT-0760

SHEET

68 OF

REV

Mechanics of instr Switch



3.3.3 Common Data Base References

A number of items in the common data base are referenced by the use of CDBDD - Prescribed but locally-defined (and therefore purgeable) symbols for word displacements and bit-layout parameters. We shall give here only the root symbols and prefix of items used, omitting the functional attribute suffixes. A table of Global Parameters used was given at the start of Section 3.3.

3.3.3.1 Reference to Emitter Library 2 (EL2)

The identify field (MID) is used in subroutine TRNSL8.

3.3.3.2 References to Emitter Track File (ETF)

References to the Emitter Track File are shown in the following table by subroutine.

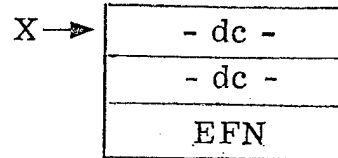
EMITTER CLASSIFICATION -1: ETF REFERENCES

Subroutine	Item Mnemonic	Item Description	Type
ECST1	EPMP	Peak Amplitude	Field
SCTCOM	ESIN	State Indicator	Flag
	ESTY	Scan Type	Field
	ESPD	Scan Period	Field
ECLV1	ELP	Long Pulse	Flag
	EFV	Frequency Validity	"
	EPIV	PRI Validity	"
	EPWV	PW Validity	"
	EFRQ	Frequency	Field
	EAPI	Average PRI	"
	EPW	Pulse Width	"
	EID	Identification	"
	EDIS	Display Code	"

3.4 INPUT/OUTPUT FORMATS

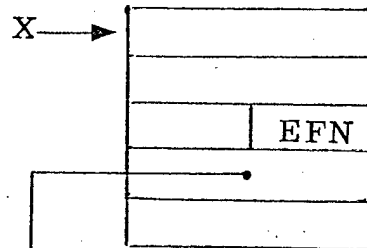
3.4.1 ECNR - - Driver

Inputs: On input X-Reg points to (contains the address of, abbreviated: X→) a three word block of which the = the number of the emitter track file to be classified. Symbolically:

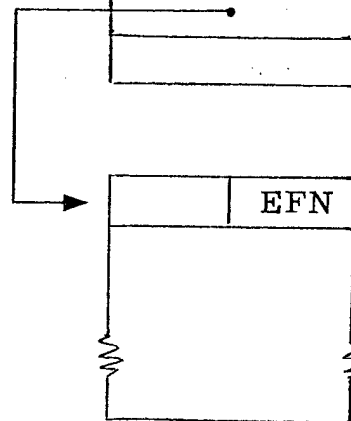


(dc = don't care)

Outputs: The output of ECNR is accomplished by a call on EXMES before ECNR returns to its caller. At the time of this call



Five-word block containing message
See Section 3.3.1.4 for internal format.



The 4th word of the message points to a 1 to 17-word Candidate List. See 3.3.1.1 for internal format.

Other outputs include permanent changes made to ETF (EFN). These will be shown in the subroutines where they are made.

3.4.2 ECLV1 (Subroutine of ECDR)

Inputs: X = EFN $0 \leq \text{EFN} \leq 127$

AND the emitter track file ETF (EFN) at address =
ETF + 16 * EFN to which this refers.

Outputs: Directly in ECLV1 Proper

EFN planted in 1st word (header) of Candidate List.

If no Candidate List (Failure):

- Failure codes in ETF ident field (EID)
- Display code in ETF display code field (EDIS)

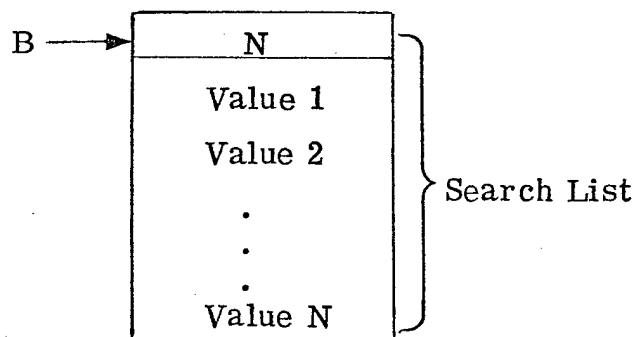
Fail/no fail indication by return to Call +1/Call +2
respectively. In either case, X → Candidate List.

Indirectly in Subroutines

Create balance of Candidate List.

3.4.2.1 BIRCH (Subroutine of ECLV1)

Inputs: A = Search Key



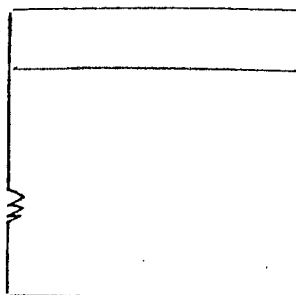
Outputs: B = Index L such that

$$\text{Value}_L \leq \text{Search Key} < \text{Value}_{L+1}$$

3.4.2.2 TRNSL8 (Subroutine of ECLV1)

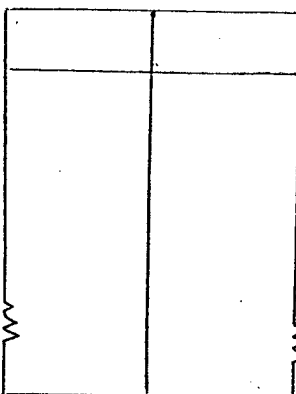
Inputs:

A →



Candidate List in KWC/BV
format 2-17 words long.
See Section 3.3.1.2.2 for
format description.

B →



33-Words area for storing
translated Candidate List.
See Section 3.3.1.1 for
internal format.

Outputs:

Translated list stored as directed.

3.4.2.3 Insect (Subroutine of ECLV1 and TRNSL8)

Inputs: A →

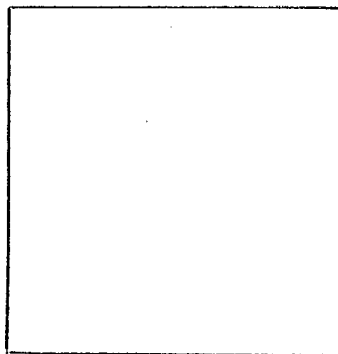
ADDR (List 1)

ADDR (List 2)

ADDR (List 3)

List i is a 2 to 17-
word list in KWC/BV
format.
See Section 3.3.1.2.2.

B →

17-word block for
storing intersection
result

E =

Switch Instruction

= NOP

For 2-way intersection
(ADDR (List 3) not used)

= JMP \$ + 2

For 3-way intersection

Outputs: Intersection NullA-Reg = \emptyset and return to Call +1Intersection Not NullA-Reg $\neq \emptyset$ (= keyword of intersection)Intersection stored as directed in KWC/BV format
(Section 3.3.1.2.2).3.4.2.3.1 Parity (Subroutine of INSECT)Inputs: A-Reg = ArgumentOutputs: A-Reg = # of 1's bits in input argument.

RAYTHEON

RAYTHEON COMPANY
LEXINGTON, MASS. 02173

CODE IDENT NO.

49956

SPEC NO.

53959-GT-0760

SHEET

74 OF

REV

3.4.3 ECST1 (Subroutine of ECDR)

Inputs: X = EFN

Outputs: If peak amplitude > amplitude threshold
(E PMP) (AMTHR)

Return to Call +2, no changes to ETF.

Else call SCTCOM and return to Call +1.

3.4.3.1 SCTCOM (Subroutine of ECST1 and ECST2)

Inputs: X = Addr ETF (EFN) = ETF + 16 * EFN

Output: (X-unchanged)

Complement ETF State Indicator (ESIN)

If ESIN now = \emptyset return, else

Set ETF Scan Type (ESTY) = Circular (CIRC)

Set ETF Scan Period (ESPD) = Time-out (TOUT)

Return

RAYTHEON

RAYTHEON COMPANY
LEXINGTON, MASS. 02173

CODE IDENT NO.

49956

SPEC NO.

53959-GT-0760

SHEET

75 OF 89

REV

3.5

REQUIRED EXTERNAL SUBROUTINES

Only one subroutine external to this nest is called. This is
EXMES

To return a message to the Executive, see Section 3.4.1.

RAYTHEON

RAYTHEON COMPANY
LEXINGTON, MASS. 02173

CODE IDENT NO.

49956

SPEC NO.

53959-GT-0760

SHEET

76 OF 89

REV

3.6 CONDITIONS FOR INITIATION

This subprogram shall have unconditional entry and shall require no special initialization procedure.

RAYTHEONRAYTHEON COMPANY
LEXINGTON, MASS. 02173

CODE IDENT NO.

49956

SPEC NO.

53959-GT-0760

SHEET

77 OF 89

REV

3.7 SUBPROGRAM LIMITATIONS

3.7.1 Timing

No timing sensitivities or limitations.

3.7.2 Algorithm Limits

The present design is based on the KWC/BV format which is limited to the representation of trunk lists 256 in length. To go beyond 256 would require a second keyword (taking the limit up to 512) and this is not a trivial modification.

3.7.3 Error Testing/Reasonableness Checks

None.

RAYTHEON

RAYTHEON COMPANY
LEXINGTON, MASS. 02173

CODE IDENT NO.

49956

SPEC NO.

53959-GT-0760

SHEET
78 OF

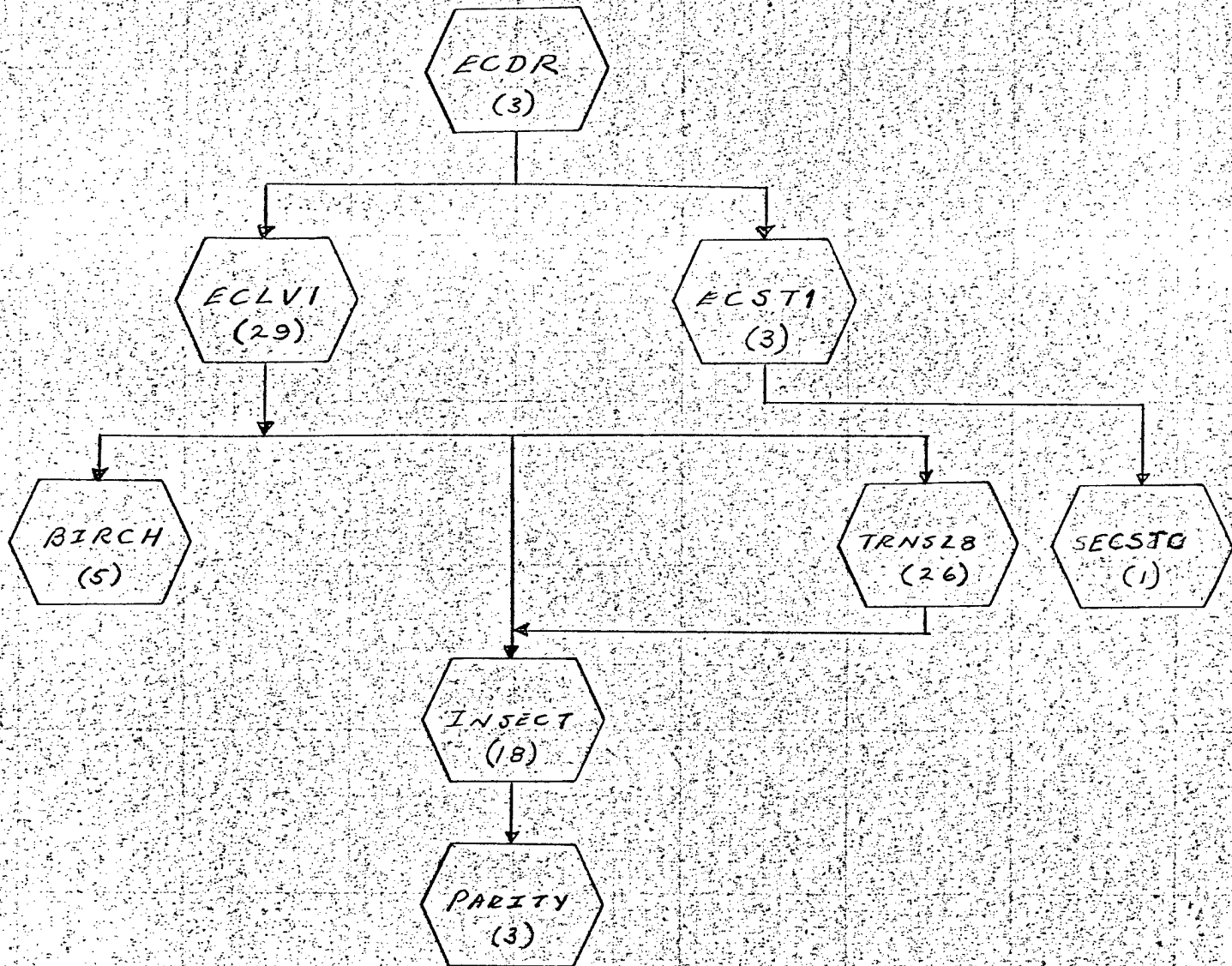
REV

3.8

INTERFACE DESCRIPTION

3.8.4 EMITTER CLASSIFICATION PROCESSING - 1

WHO CALLS WHOM



NUMBERS IN PARENTHESES = MAXIMUM STACK DEPTH INCREASE CAUSED BY MODULE. THIS INCLUDES THE ENTRY RETURN ADDRESS, BUT NOT ADDITIONS TO STACK BY DEPENDENT SUBROUTINES.

TOTAL MAXIMUM = 79 (IN PARITY WHEN INSECT CALLED BY TRANSL8)

RAYTHEON

RAYTHEON COMPANY
LEXINGTON, MASS. 02173

CODE IDENT NO.

49956

SPEC. NO.

53959-GT-0760

SHEET

80 OF

REV

APPENDIX

Detailed Examples of Algorithms

- A. Binary Search
- B. Intersection
- C. Parity (1's Bit Counting)

APPENDIXDetailed Examples of Algorithms

This appendix presents instruction-by-instruction trace of three of the algorithms. The lines are keyed to subroutine internal labels or internal, unlabeled sections (; N). ALL numbers are hexadecimal (Base 16).

A. Binary Search (BIRCH)

Entry Conditions: _____

A-Reg = 0 0 2 7

B-Reg = 3 0 0 0 :
 3 0 0 1
 3 0 0 2
 3 0 0 3
 3 0 0 4
 3 0 0 5
 3 0 0 6
 3 0 0 7
 3 0 0 8

0	0	0	8
0	0	0	0
0	0	0	5
0	0	1	0
0	0	1	5
0	0	2	0
0	0	2	5
0	0	3	0
0	1	0	0

List to Follow:
8 long

S-Reg = 1 0 3 0 before Call

RAYTHEON

RAYTHEON COMPANY
LEXINGTON, MASS. 02173

CODE IDENT NO.

49956

SPEC NO.

53959-GT-0760

SHEET

82 OF

REV

(1 of 2)

Binary Search

Appendix A -

Instruction	A	B	E	X	S	Remarks
BIRCH: PSH A	0	0	e	x	1	(1 0 2 E) = 0 0 2 7
PSH X	.	0	e	x	1	(1 0 2 D) = x x x x
PSH E	1	(1 0 2 C) = e e e e
CLR X	.	.	.	0	.	F F F F = -1
COM X, X	.	.	.	0	.	
LDN, E	.	.	.	F	.	
B, INC	.	3	F	.	.	
LDC -1, B	F	0	F	.	1	(1 0 2 B) = 3 0 0 1
PSH B	F	
LOOPG: SAD, X X	.	3	F	.	.	JMP Comon
COMON: SUB, B E	.	0	F	.	.	0 0 2 7 vs. 0 0 2 0
MOV A, E	.	5	F	.	.	JEZ, JNG Not Taken
SAD, X X	.	.	F	.	.	No Skip
(; 7) LDS 3, S	0	.	F	.	.	No Skip
XOR 0, B	0	.	F	.	.	
LDS 3, S	0	.	F	.	.	
CSL 0, B	.	.	F	.	.	
GRTR: MOV E, A	F	.	F	.	.	
ISZ, E E	.	.	F	.	.	
LOOPG: SAD, X X	.	3	F	.	.	JMP Comon
COMON: SUB, B E	.	0	F	.	.	0 0 2 7 vs. 0 0 3 0
MOV A, E	.	.	F	.	.	JEZ JNG Not taken
SAD, X X	.	.	F	.	.	Skip taken
(; 7) LDS 3, S	0	.	F	.	.	No Skip
XOR 0, B	0	.	F	.	.	
LDS 3, S	0	.	F	.	.	
CSL 0, B	.	.	F	.	.	
LESS: MOV E, A	F	.	F	.	.	
IS2, E E	.	.	F	.	.	

RAYTHEON

RAYTHEON COMPANY
LEXINGTON, MASS. 02173

CODE IDENT NO.

49956

SPEC NO.

53959-GT-0760

SHEET
83 OF

REV

Appendix A - Binary Search (continued)

(2 of 2)

Instruction	A	B	E	X	S	Remarks
LOOP: SAD, XX NEG E, E	.	.	F F F	.	.	
COMON: SUB, B E MOV A, E	.	3 0 0 0	F F 0	.	.	
SAD, XX	.	.	F F F D	.	.	
(; 7) LDS 3, S	0 2 7	.	F F F E	.	.	0 0 2 7 vs. 0 0 2 5 JEZ, JNG Not taken
XOR 0, B	0 0 2	No Skip
LDS 3, S	0 0 2 7	No Skip
CSL 0, B	
GRTR: MOV E, A	F F F E	.	F F F F	.	.	
ISZ, E E	
LOOP: SAD, XX	.	3 0 0 7	F F F F	.	.	JMP Comon
COMON: SUB, B E MOV A, E	.	.	F F F E	.	.	
SAD, XX	.	.	F F F F	.	.	
(; 7) LDS 3, S	0 2 7	0 0 2 7 vs. 0 0 3 0 JEZ, JNG Not taken
XOR 0, B	0 0 1 7	Skip taken
LDS 3, S	0 0 2 7	Skip taken
CSL 0, B	
LESS: MOV E, A	F F F F	.	0 0 0 0	.	.	
ISZ, E E	
DSZ, B B	.	3 0 0 6	.	.	.	
DONE: SUB, B	F F F F	0 0 0 5	0 0 0 0	F F F F	1 0 0 0 0 0	Result in B-Reg
S, INC	.	.	e e e e	.	.	
POP E	.	.	.	x x x x	1 1 1 1 1	
POP X	
POP A	0 0 2 7	
EXIT	0 0 2 7	0 0 0 5	e e e e	x x x x	1 1 1 1 1	

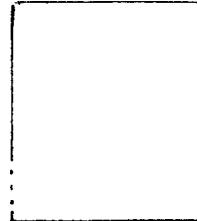
B. Intersection Routine (INSECT)

Entry Conditions:

A = 2 5 A B:

3	A	4	3
4	1	B	E

B = 3 1 2 2:



17 Empty Locations

E = ∅ ∅ ∅ ∅ (NOP)

3 A 4 3:

1	0	0	0	①	0	0	1	①	0	1	0	0	0	0	0
1	1	0	0	1	1	0	0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	1	0	1	0	0	0	0	0	1	0	1	0	1	0

KW1

KWC/BV Lists

4 1 B E:

0	0	0	1	①	0	0	0	①	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	1	1	0	0	1	0	1
1	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

KW2

S = 1 1 ∅ 4

These arrows show the pairs of BV words that must be ANDed. They correspond to the circled 1's in the keywords.

RAYTHEON

RAYTHEON COMPANY

LEXINGTON, MASS. 02173

SPEC NO.
53959-GT-0760

SHEET

85 OF

REV

(1 of 4)

Intersection Algorithm -1

Appendix B

Instruction	A	B	E	X	S	Remarks
INSECT: PSH X	2 5 A B	3 1 2 2	0 0 0 0	x x x x	1 1 0 2	(1 1 0 2) = x x x x
PSH E	1 1 0 1	(1 1 0 1) = 0 0 0 0 (NOP)
PSH B	1 1 0 0	(1 1 0 0) = 3 1 2 2
(;1) LDS, B A, INC	2 5 A C	3 A 4 3	.	.	1 1 0 0	B ← 1st List Addr
PSH B	1 0 0 F	(1 0 0 F) = 3 A 4 3
LDS, B A, INC	2 5 A D	4 1 B E	.	.	1 0 0 F	B ← 2nd List Addr
PSH B	1 0 0 E	(1 0 0 E) = 4 1 B E
LDS, B A, INC	2 5 A E	J U N K	.	.	1 0 0 F	B ← 3rd List Addr = Junk
PSH B	1 0 0 D	(1 0 0 D) = Junk
MOV S, X	.	.	.	1 0 F D	1 0 0 F	Make Room Temp Storage
SUB, S=X'B'	0 0 0 0	.	.	.	1 0 0 F	Head of Double Words
(;2) CLR A	.	F F F F	.	.	.	Proto KWI
COM A, B	(EA) = (1 1 0 L) = NOP
(;3) EXQ X'F', S	Avoid 3rd List
JMP INS01	
INS01: ISZ, X X	
LDS, E X, IND	.	.	1 8 8 0	1 0 F F	.	((1 0 F E)) = (4 1 B E) = KW2
STD 4, S	.	.	.	1 0 F F	.	(1 0 F F 6, F) = (0, 1 8 8 0)
AND, B E	.	1 8 8 0	.	.	.	
LDS, E X, IND	.	.	8 9 A 0	1 1 0 0	.	((1 0 F F)) = (3 A 4 3) = KW1
STD 2, S	(1 0 F F 4, 5) = (0, 8 9 A 0)
AND, B E	.	0 8 8 0	0 8 8 0	.	.	JE2, B Not taken
(;6) +1 MOV B, E	
STD 0, S	(1 0 F F 2, 3) = (0, 0 8 8 0)
STS 8, S	(1 0 F F A) = 0 0 0 0
LDS, A = 16	0 0 1 0	
STS 9, S	(1 0 F F B) = 0 0 1 0
INLOOP: LDS, X=1	0 0 0 0	.	0 8 8 0	0 0 0 1	.	
LDD 0, S	0 0 0 0	
INSHFT: JNG, E	0 0 0 0	.	1 1 0 0	.	.	
(= INS02)						
SAD =1						Not Taken

RAYTHEON

RAYTHEON COMPANY

LEXINGTON, MASS. 02173

CODE IDENT NO.

49956

SPEC NO.

53959-GT-0760

SHEET

87 OF

REV

Appendix B Intersection Algorithm -1 (continued) (3 of 4)

Instruction	A	B	E	X	S	Remarks
INS04: MOV A, E	.	.	8 8 0 4	.	.	A=Non-Zero Result Word (1 0 F A)=0 0 0 1
ISZ 8, S	0 0 1 2 3	
LDS 8, S	3 1 2 4	
ADD X'E', S	3 1 2 4	
STS, E A, INC	0 0 1 0 0 B	Result Destination Addr (3 1 2 3)=8 8 0 4 *
INS05: LDS 9, S	0 0 0 0 0	Tests for Done
SUB, A X	0 0 0 0 0	
JE2 INDUN2	Not Taken
STS 9, S	0 0 0 0 1	.	1 0 0 0 0	.	.	(1 0 F B) = 0 0 0 0 B
LDD 0, S	Taken
JNZ, E (=INLOOP)	
INLOOP: LDS, X =1	0 0 1 1	.	0 0 0 0	0 0 0 0	1 1 4	Omitting Shift Loop Detail
INS02: SAD =1	0 0 1 1	.	3 4 0 0 0	.	.	(1 0 F 2, 3) = (11, 0)
STD 0, S	0 0 1 1 3	
(;11) LDD 2, S	0 0 1 1 3	
SAD, A X	(1 0 F 4, 5)=(0 1 1 4, 4 0 0 0)
STD 2, S	
JSB PARITY	0 0 0 4	
ADD X'D', S	3 A 4 0 0	
LDS, A A, INC	0 0 0 0 1	(1 0 F C)=0 0 0 1
STS X'A', S	0 0 0 3 1	.	0 0 0 0	.	.	
(;12) LDD 4, 5	0 0 0 3 1	.	0 0 0 0	.	.	
SAD, A X	0 0 0 3 1	.	0 0 0 0	.	.	(1 0 F 6, 7) = (31, 0)
STD 4, 5	0 0 0 3 1	.	0 0 0 0	.	.	
JSB PARITY	0 0 0 3 1	.	0 0 0 0	.	.	
ADD X'C', S	0 4 1 0 0 0	
LDS, A A, INC	0 0 0 0 0	
AND X'A', S	0 0 0 0 0	0 0 0 1 ^ C 0 0 0
(;13) EXQ X'F', S	0 0 0 0 0	A NOP
JMP INS03	Avoid 3rd List
INS03: JNZ INS04	Not Taken

RAYTHEON

RAYTHEON COMPANY
LEXINGTON, MASS. 02173

CODE IDENT NO.

49956

SPEC NO.

53959-GT-0760

SHEET

88 OF

REV

(4 of 4)

Intersection Algorithm -1 (continued)

Appendix B

Instruction	A	B	E	X	S	Remarks
(;16) LDS \emptyset , S	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	Head of DWI
XOR = 1	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	(1 \emptyset F 2, 3) = (1 \emptyset , \emptyset)
STS \emptyset , S	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	Not Taken
INS \emptyset 5: LDS 9, S	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	(1 \emptyset F B) = \emptyset \emptyset \emptyset F
SUB, A X	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	Not Taken
JE2. INDUN2	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	Not Taken
STS 9, S	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	Complete Shifts
LDD \emptyset , S	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	(EA) = (3 1 2 2) = \emptyset 8 \emptyset \emptyset *
JNZ, E (=INLOOP)	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	
(;19) SAD 9, S	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	
INDUN1: STS	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	
X'E', SI	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	
ADD, S = X'1 \emptyset '	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	
POP X	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	
EXIT	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	

RAYTHEON

RAYTHEON COMPANY
LEXINGTON, MASS. 02173

CODE IDENT NO.

49956

SPEC NO.

53959-GT-0760

SHEET
89 OF 89

REV

Appendix C

Parity - - Counting One's Bits

Entry Conditions:

A-Reg = 8 0 0 1

S-Reg = 2 0 1 4

Instruction	A	B	E	X	S	Remarks
PARITY:						
JEZ DONE	8 0 0 1	b b b b	e e e e	.	2 0 1 3	Not Taken
PSH B	2 0 1 2	(2 0 1 2)=b b b b
PSH E	2 0 1 1	(2 0 1 1)=e e e e
CLR B	.	0 0 0 0	.	.	.	
PLOOP:						
MOV A, E	.	.	8 0 0 1	.	.	No Skip
DSZ, E E	.	.	8 0 0 0	.	.	
NOP	
AND, A E	8 0 0 0	0 0 0 1	.	.	.	Taken
ISZ, B B	
JNZ PLOOP	
PLOOP:						
MOV A, E	.	.	8 0 0 0	.	.	No Skip
DSZ, E E	.	.	7 0 0 F	.	.	
NOP	
AND, A E	0 0 0 0	0 0 0 2	.	.	.	Not Taken
ISZ, B B	
JNZ PLOOP	
(;3) MOV B, A	0 0 0 2	.	e e e e	.	2 0 1 2	
POP E	2 0 1 3	
POP B	0 0 0 2	b b b b	e e e e	.	2 0 1 4	
DONE: EXIT						